Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

РАЗРАБОТКА СЕТЕВОЙ ИГРЫ «МАФИЯ»

БГУИР КР 6 - 05 - 06 12 01 013 ПЗ

Выполнила: студент группы 310901 Лазаревич М. Д.

Проверила: Болтак С. В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc199237237)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc199237238)

[1.1 Обзор аналогов 6](#_Toc199237239)

[1.2 Постановка задачи 8](#_Toc199237240)

[2 ПРОЕКТИРОВАНИЕ ЗАДАЧИ. 9](#_Toc199237241)

[2.1 Структура программы 9](#_Toc199237242)

[2.2 Проектирование интерфейса программного средства 10](#_Toc199237243)

[2.3 Проектирование функционала программного средства 16](#_Toc199237247)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 20](#_Toc199237248)

[3.1 Взаимодействие клиентов с сервером 20](#_Toc199237249)

[3.2 Подключение к игре 21](#_Toc199237250)

[3.3 Ночная фаза 22](#_Toc199237251)

[3.4 Дневная фаза 24](#_Toc199237252)

[3.5 Завершение игры 25](#_Toc199237253)

[4 ТЕСТИРОВАНИЕ 27](#_Toc199237254)

[5 ПРИМЕНЕНИЕ ПРОГРАММЫ 30](#_Toc199237255)

[5.1 Интерфейс программного средства 30](#_Toc199237256)

[ЗАКЛЮЧЕНИЕ 34](#_Toc199237257)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 35](#_Toc199237258)

[ПРИЛОЖЕНИЕ А 36](#_Toc199237259)

# **ВВЕДЕНИЕ**

Интернет и компьютерные технологии давно стали неотъемлемой частью современной жизни. С их помощью люди не только общаются и узнают новости, но и находят новые способы проведения досуга. Одним из таких способов являются сетевые игры, которые объединяют игроков со всего мира, позволяя им взаимодействовать и соревноваться в режиме реального времени.

Сетевые игры представляют собой сложные программные продукты, требующие применения современных технологий и глубоких знаний в области программирования. Интерес к разработке таких игр растёт, благодаря развитию языков программирования и инструментов, которые упрощают создание интерактивных приложений.

Одной из самых популярных игр, которая сохраняет свою актуальность на протяжении нескольких десятилетий, является «Мафия». «Мафия» — клубная командная психологическая пошаговая ролевая игра с детективным сюжетом, моделирующая противостояние двух групп: мафии и мирных жителей. Благодаря своей увлекательной механике, она получила множество вариаций, включая сетевые версии.

Курсовая работа посвящена созданию сетевой игры «Мафия», которая предоставляет игрокам возможность взаимодействовать в режиме реального времени через удобный и интуитивно понятный интерфейс, с использованием языка программирования *Python*[2]. В рамках работы предполагается реализовать базовый игровой функционал, включая распределение ролей, управление игровыми фазами (день и ночь), возможность общения между игроками и голосование.

Разработка сетевой версии данной игры является актуальной задачей, так как она объединяет игровой процесс с современными технологиями. *Python* выбран из-за своей простоты, широких возможностей и наличия библиотек, подходящих для разработки сетевых приложений, таких как *websocket*[3] для сетевого взаимодействия и *Flet*[4] для создания игрового интерфейса.

# **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **Обзор аналогов**

Сегодня существует множество реализаций сетевой игры «Мафия». Все они предоставляют стандартный набор функций, таких как создание комнат, обработка взаимодействия между игроками, управление игровым процессом. Рассмотрим 2 популярные платформы – «*Polemica Game*» и «Мафия Онлайн».

«*Polemica Game*» – это платформа, которая сочетает элементы классической «Мафии» с современными игровыми механиками[5]. Она ориентирована как на любителей, так и на профессионалов, предлагая несколько режимов игры с различной сложностью и динамикой. Внешний вид сервиса представлен на рисунке 1.1.

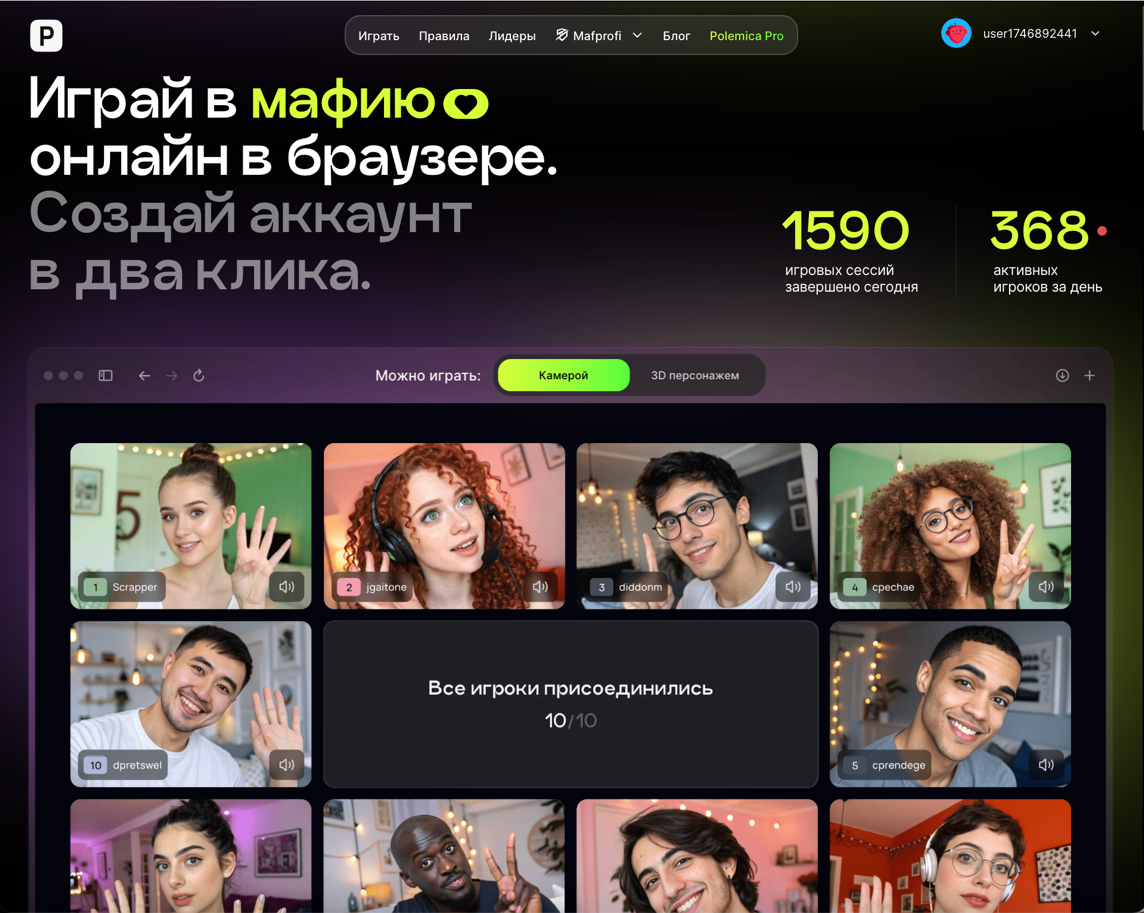


Рисунок 1.1 – Платформа «*Polemica Game*»

Основная особенность данной платформы – это наличие разнообразных режимов игры:

* «Классическая мафия» – режим для 10 игроков с системой начисления очков за каждый ход;
* «Угадайка» – быстрый и интуитивный режим для 4 игроков, где важно принимать решения в условиях ограниченной информации;
* «Один на один» – инновационный формат, где мирный игрок должен вычислить мафию среди двух оппонентов.

Платформа автоматически распределяет роли, сменяет фазы дня и ночи, отслеживает время для речи и голосований. Играть можно как на персональном компьютере, так и на смартфоне. Для большего погружения можно использовать микрофон и камеру. Также можно участвовать в игре в качестве наблюдателя.

Важной частью игры «Мафия» является общение между игроками. На данной платформе активно продвигаются здоровая коммуникация и уважение к участникам, а новичкам помогают разобраться в правилах и механиках игры. За нарушение этикета выдаются штрафы.

Помимо основной механики игры, «*Polemica Game*» также предоставляет возможность участвовать в киберспортивных соревнованиях и турнирах, где победители могут выиграть крупные призы, а интеграция со сторонними сервисами, например YouTube, позволяет наблюдать за ходом игр.

Второй аналог – «Мафия Онлайн». Это проект, который представляет собой реализацию классической игры «Мафия» с доступом через браузер[6]. Интерфейс платформы представлен на рисунке 1.2.

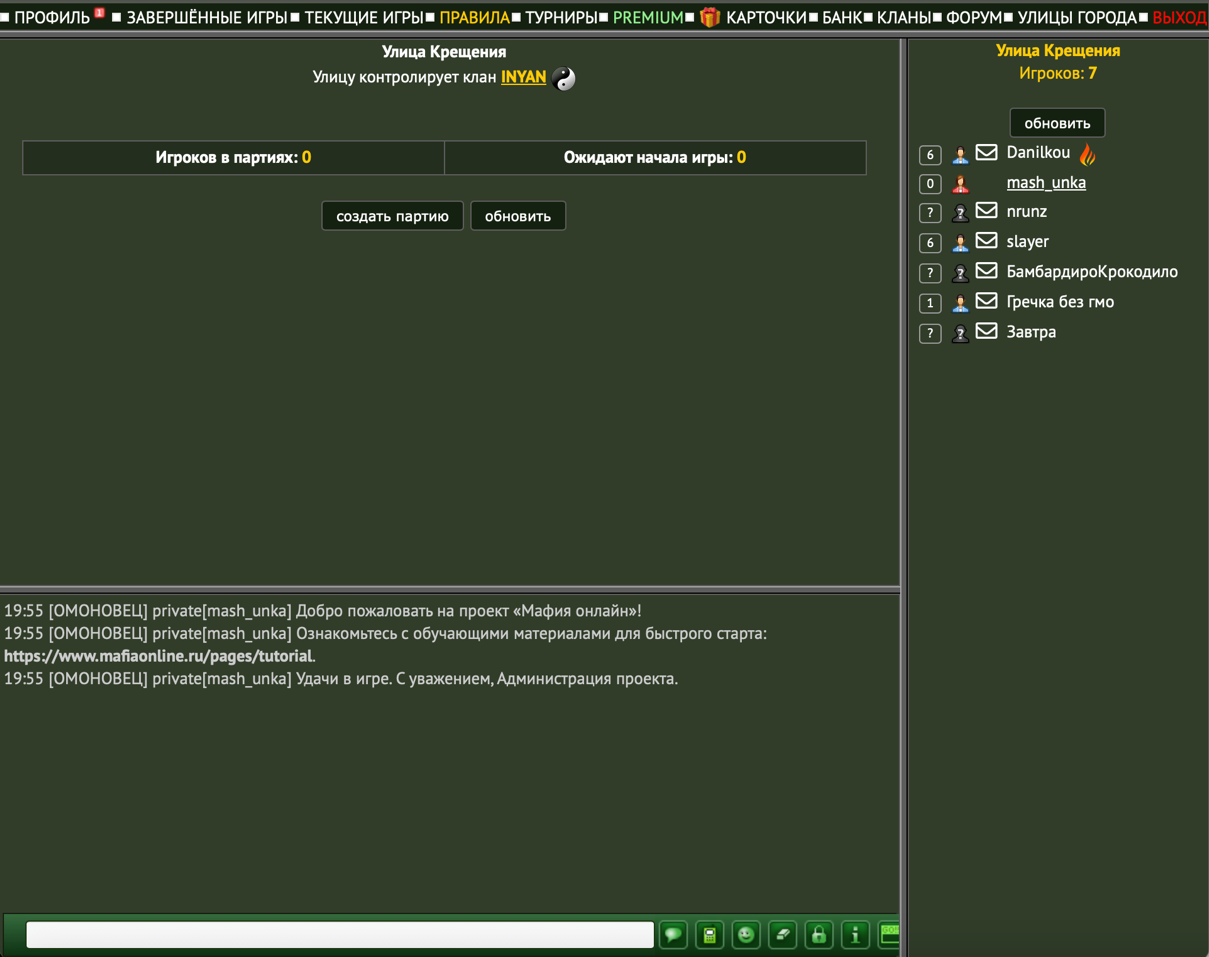


Рисунок 1.2 – Интерфейс платформы «Мафия Онлайн»

Данная платформа представляет собой достаточно простую реализацию игровой механики. В игре участвуют роли мафии, детектива, мирных жителей и других персонажей. Все роли назначаются случайным образом, что позволяет каждому раунду быть уникальным.

Платформа также активно поддерживает общение между игроками, включая форумы, блоги и обучающие материалы.

## **Постановка задачи**

В рамках данной курсовой работы необходимо разработать сетевую игру «Мафия». В перечень задач при разработке игры входит:

* изучить правила и механики классической игры «Мафия»;
* провести анализ существующих реализаций игры в онлайн-формате;
* реализовать игровой интерфейс;
* определить функциональные требования к игре.

Основные функциональные требования игры включают реализацию механики игры:

* создание игровой комнаты;
* распределение ролей (мафия, детектив, мирные жители);
* смена фаз игры (день, ночь);
* голосование и подсчёт результатов;
* проверка условий победы;
* реализация завершения игры.

Помимо основных функций игрового процесса необходимо реализовать интуитивно понятный и удобный интерфейс:

* основной игровой интерфейс, отображающий текущую фазу, роль игрока и его функции;
* интерфейс, отвечающий за отображение процессов на сервере, например, ожидание подключения, подсчета голосов;
* экран завершения игры с победителями и проигравшими.

За автоматизацию игровых процессов будет отвечать сервер, который реализует следующие функции:

* подключение нескольких игроков к одной игровой комнате;
* синхронизация игровых данных между клиентами и сервером;
* обработка игровых событий.

# **ПРОЕКТИРОВАНИЕ ЗАДАЧИ**.

## **Структура программы**

При разработке программы использовано 3 основных модуля:

* *Interface* – модуль, отвечающий за обновления интерфейса в зависимости от фазы и этапа игры;
* *Network* – модуль, который отвечает за сетевое взаимодействие клиента с сервером;
* *Game* – модуль, который отвечает за взаимодействие сервера с клиентом.

В основных модулях реализованы 4 класса: *Game*, *NetworkClient*, *Player*, *GameUI* [7]. Класс *Game* отвечает за ход игры и реализует логику игровых событий, таких как ночная и дневная фазы, голосование, подсчет результатов. Класс *Player* реализует игрока с именем, ролью и статусом. Класс *NetworkClient* реализует игрока как клиента сервера, а также отвечает за механизм взаимодействия игрока с процессом игры, принимая сообщения от сервера. Класс *GameUI* отвечает за смену интерфейса в зависимости от роли игрока, фазы и этапа игры. Методы и поля классов, описанных выше, представлены в таблице 2.1.

Таблица 2.1

|  |  |  |
| --- | --- | --- |
| 1.Название | 2. Поля | 3. Методы |
| *Game* | self.players, self.connections, self.game\_started, self.phase, self.day\_number, self.night\_actions, self.votes, self.min\_players, self.start\_task, self.lock. | handle\_connection(), handle\_message(), handle\_disconnect(), start\_game(), game\_loop(), night\_phase(), wait\_for\_night\_action(), night\_action\_received(), handle\_night\_action(), process\_night\_actions(), day\_phase(), collect\_votes(), handle\_vote(),end\_game(), process\_votes(), check\_win\_conditions(), send\_to\_player(), broadcast() |
| *Player* | self.name, self.role,  self.is\_alive |  |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| 1. | 2. | 3. |
| *NetworkClient* | self.page, self.ws, self.name, self.role, self.players, self.game\_ui, self.nightdata | connect(), listen(), send(), send\_action(), handle\_message(), handle\_role\_assignment(), handle\_night\_action\_request(), handle\_phase\_change(), handle\_players\_update(), handle\_game\_over(), handle\_error(), handle\_disconnect(), close(), set\_game\_ui() |
| *GameUI* | self.page, self.network,  self.player\_name, self.connect\_btn, self.role\_display,  self.players\_list,  self.main\_column. | show\_connect\_view(), connect(), show\_waiting\_view(), show\_role\_view(), auto\_hide\_role\_view(), show\_night\_phase(), \_create\_mafia\_interface(), \_create\_doctor\_interface(), \_send\_mafia\_choice(), \_send\_doctor\_choice(), show\_day\_phase(), show\_countdown(), clear\_page(), update\_players\_list(),  show\_game\_over() |

## **Проектирование интерфейса программного средства**

Проектирование интерфейса программного средства происходит с помощью библиотеки *Flet*. *Flet* — это современная библиотека для создания интерактивных пользовательских интерфейсов на языке *Python*.

### **Меню игры**

Меню игры отображается игроку сразу при запуске приложения. На окне с меню отображается три кнопки: «Подключиться к игре», «Правила» и «Выйти». При нажатии на первую кнопку игроку представится возможность подключиться к игровой комнате. При нажатии на кнопку «Правила» программа выведет правила игры, с которыми можно ознакомиться до начала игры. При нажатии на кнопку «Выйти», пользователь выйдет из приложения, и оно закроется. Интерфейс меню представлен на рисунке 2.1.

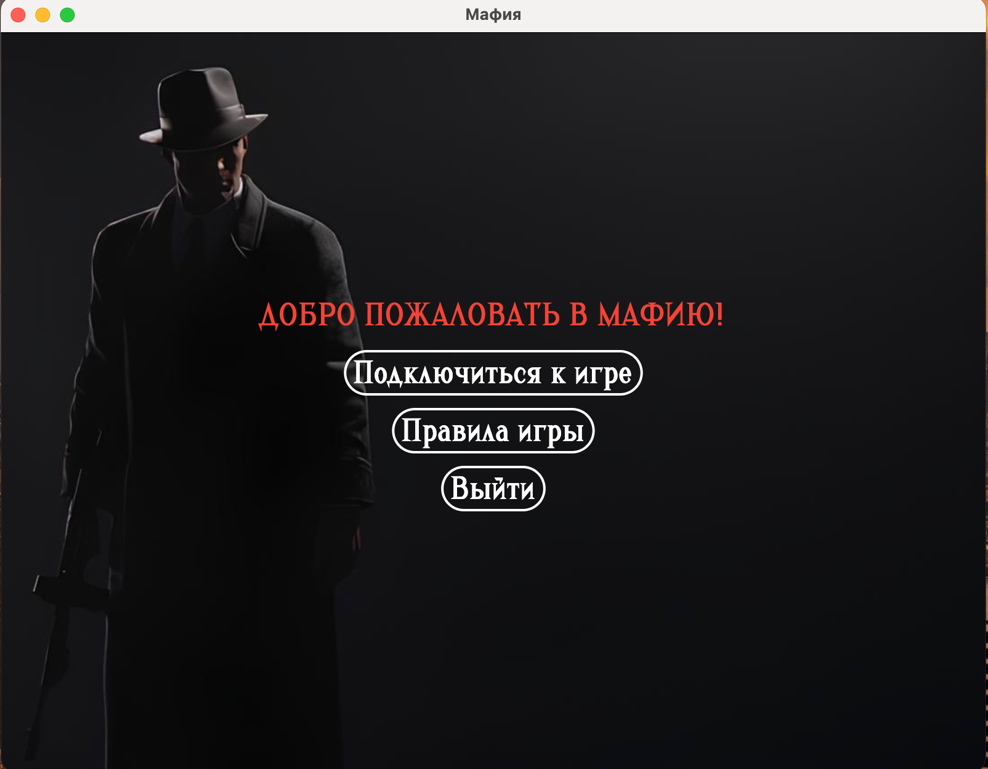


Рисунок 2.1 – Интерфейс меню игры

### **Подключение игрока к комнате**

Подключения игрока к игре происходит в несколько этапов. Сначала игроку необходимо ввести имя, под которым он хочет войти в игру. Чтобы это реализовать, игроку выводится окно подключения. На данном окне отображается 2 элемента – поле для ввода имени и кнопка «Подключится». Вид окна подключения представлен на рисунке 2.2.

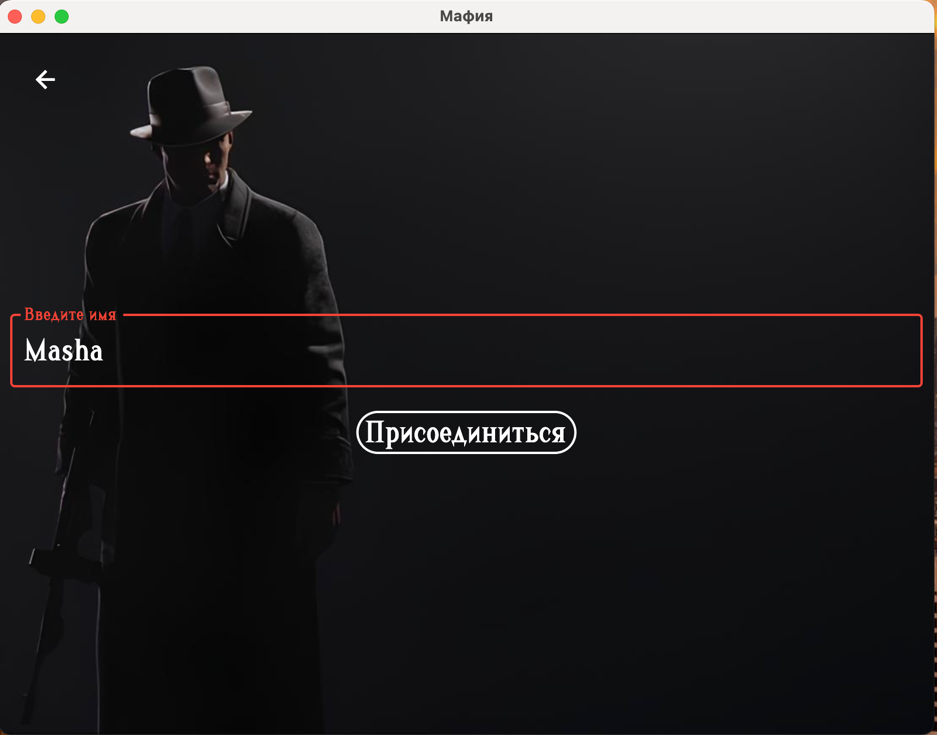


Рисунок 2.2 – Окно подключения

После подключения игрока, он попадает в фазу ожидания минимального количества игроков. Игрокам, которые уже подключились к комнате, видно, сколько игроков осталось до минимального количества, а также имена подключенных игроков. Интерфейс фазы ожидания показан на рисунке 2.3.

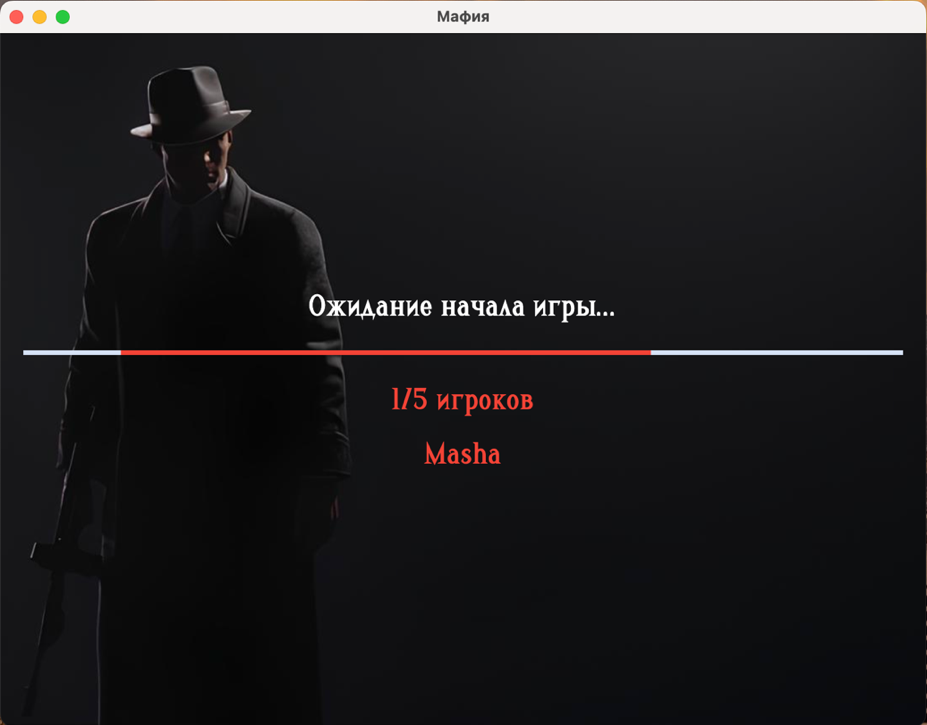


Рисунок 2.3 – Окно ожидания подключений

После того, как наберется нужное количество игроков, сервер распределяет роли и демонстрирует их игрокам, начиная обратный отсчет до начала игры. Окно с предоставленной ролью и обратным отсчетом показано на рисунке 2.4.

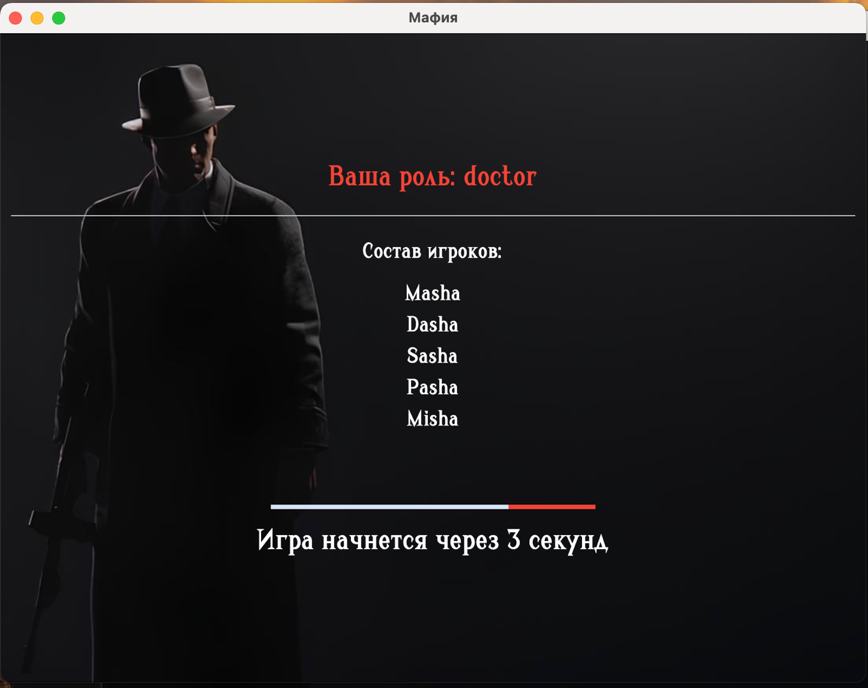


Рисунок 2.4 – Окно с предоставленной ролью и обратным отсчетом

Роли в игре:

* «Мафия» – один или несколько человек, которые тайно сотрудничают, чтобы уничтожить мирных жителей;
* «Мирные жители» – большинство участников, их цель – выявить мафию;
* «Доктор» – спасает одного игрока от мафии каждую ночь.

### **Процесс игры**

После окончания обратного отсчёта начинается первая ночь. Игроки в зависимости от роли поочередно выполняют свои действия. Сначала делает свой ход мафия, затем доктор. Каждому игроку высвечивается соответствующее окно ночной фазы с возможностью выполнить определенное действие, если это позволяет роль игрока. Интерфейс ночной фазы для мирного жителя представлен на рисунке 2.5.

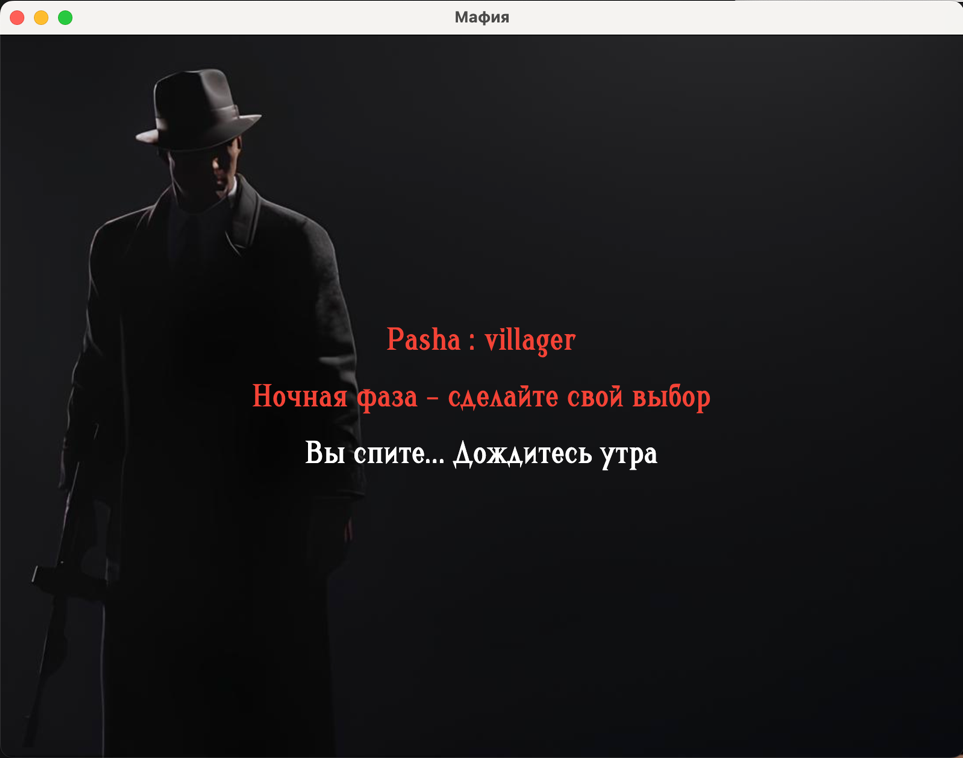


Рисунок 2.5 – Окно ночной фазы для мирного жителя

Если роль игрока – мафия или доктор, у него должна быть возможность сделать действие. Для этого на окне ночной фазы отображается выпадающий список с именами «живых» игроков и кнопка, которая отправит выбор мафии или доктора серверу. Интерфейс ночной фазы для доктора представлен на рисунке 2.6. Интерфейс ночной фазы для мафии аналогичен.

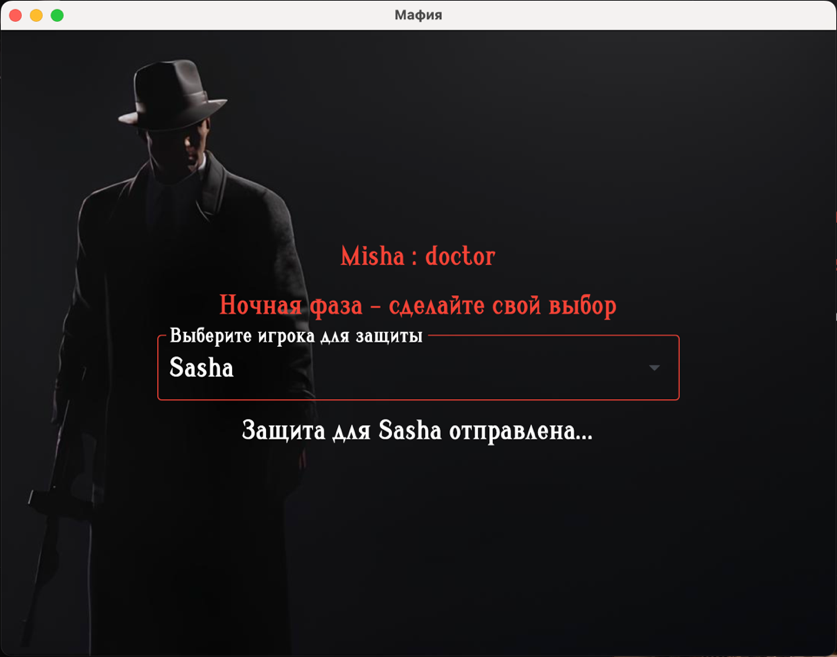


Рисунок 2.6 – Окно ночной фазы для доктора

После завершения ночи начинается дневная фаза. Она состоит из голосования и обсуждения результатов ночи, и в течение дневной фазы игроки могут переключатся с чата на окно с голосованием. Система объявляет результаты ночи – сообщает, кто был «убит» (если игрока не «спасли»), «убитый» игрок выходит из игры, а его роль сразу раскрывается.

Далее мирные жители пытаются вычислить мафию, а мафия – запутать остальных. Игроки могут выдвигать обвинения, защищаться или предлагать свои версии происходящего. Обсуждение происходит в специальном чате игроков.

Для этого есть специальное поле, в котором отображаются все сообщения, поле для ввода сообщений и кнопка отправить. В левом верхнем углу также есть кнопка «Назад», с помощью которой игрок может переключаться между чатом и окном с выбором игрока. Интерфейс чата представлен на рисунке 2.7.

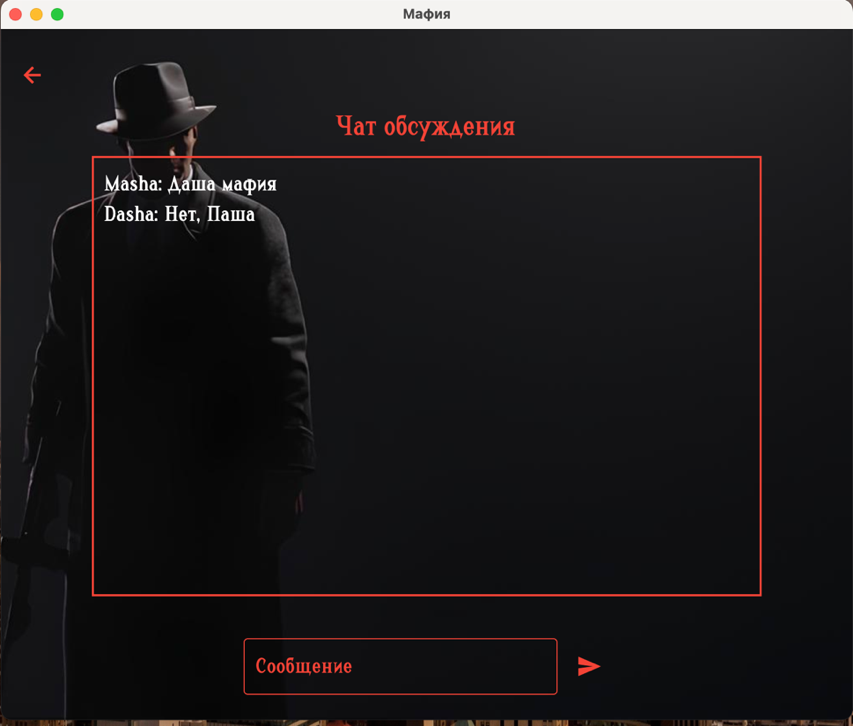


Рисунок 2.7 – Интерфейс чата

Далее каждый игрок должен указать, кого он считает мафией. В окне с голосованием представлен выпадающий список с «живыми» игроками, с помощью которого происходит голосование. Игрок, набравший большинство голосов, «выходит из игры», а его роль также раскрывается. Интерфейс дневной фазы представлен на рисунке 2.8.

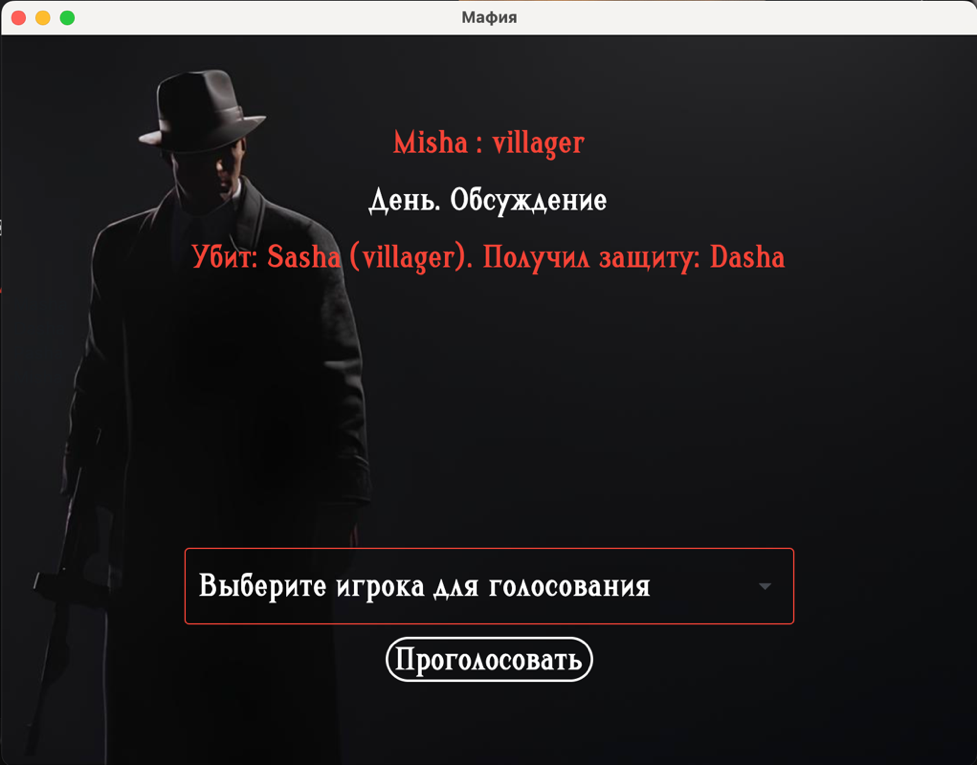


Рисунок 2.8 – Окно дневной фазы

После завершения дневной фазы, если мафия и хотя бы два мирных жителя остались в игре, начинается следующая ночь. В ходе игры будут чередоваться ночные и дневные фазы, пока не будет достигнуто одно из условий завершения игры:

* победа мирных жителей – мафия устранена в результате голосований;
* победа мафии – количество мафиози становится равно количеству мирных жителей.

По окончании игры каждому игроку выводится окно завершения игры с результатами. Интерфейс окна завершения игры представлен на рисунке 2.8.

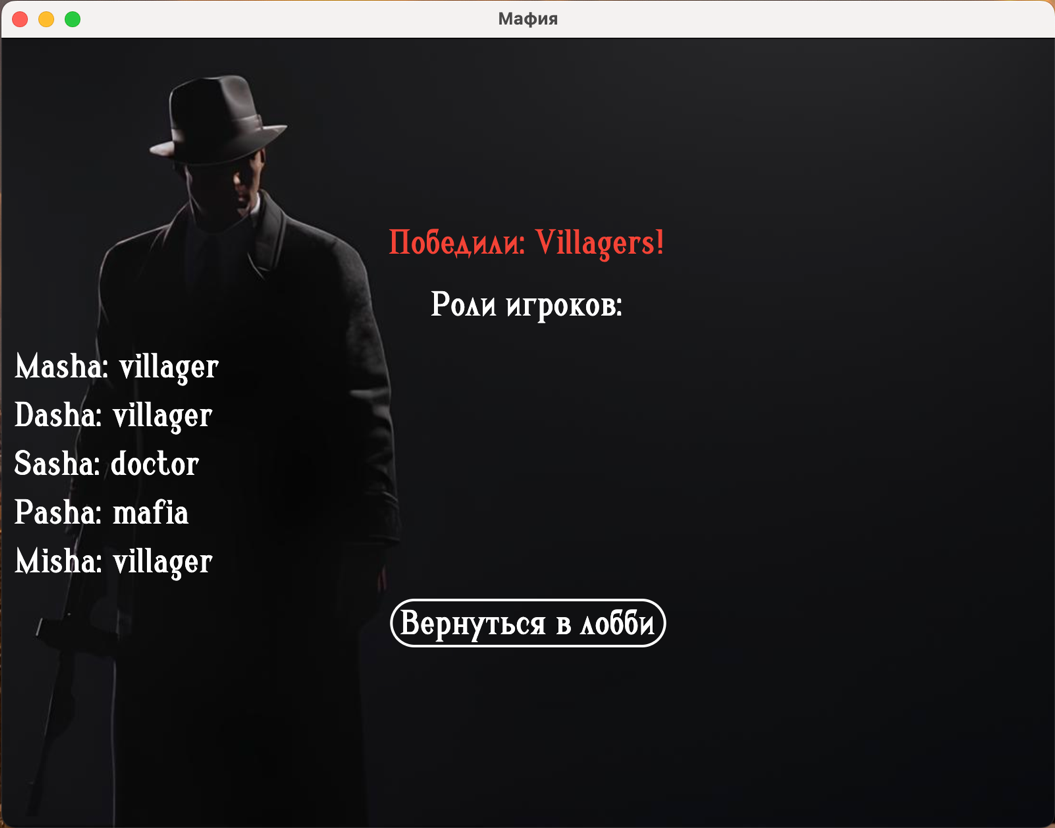


Рисунок 2.9 – Окно завершения игры

## **Проектирование функционала программного средства**

Для корректной работы программы необходимо правильно реализовать основной функционал. Для сетевой версии игры «Мафия» основными функциями является:

* организация смены фаз;
* обработка взаимодействий игроков с сервером;
* система голосования;
* обработка ночных действий игроков.

Смена игровых фаз управляется сервером с использованием асинхронной библиотеки *asyncio* для обеспечения одновременной обработки действий игроков [8]. Основный цикл игры сменяет дневную и ночную фазы между собой, пока не будет достигнуто одно из условий завершения игры. Алгоритм смены фаз отображен на блок-схеме на рисунке 2.10.

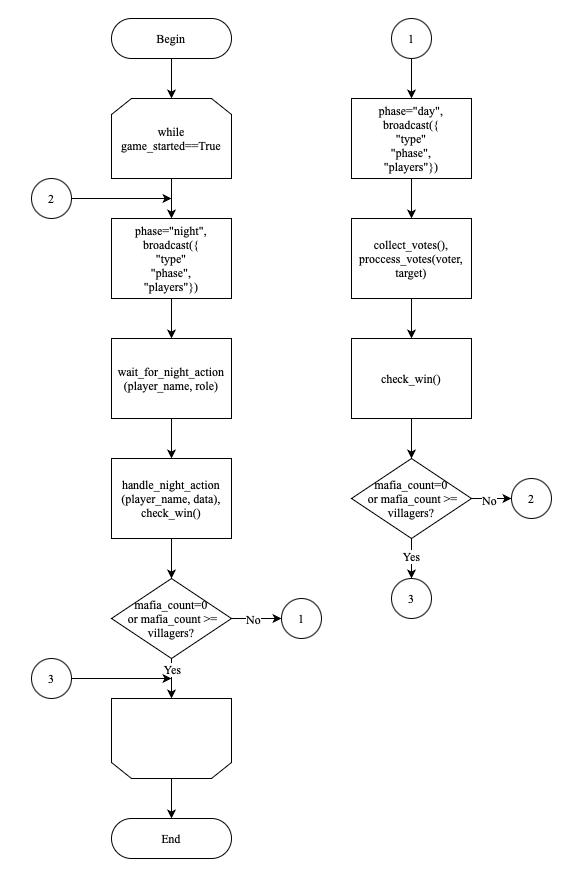


Рисунок 2.10 – Блок-схема алгоритма смены фаз

Клиенты и сервер взаимодействуют, передавая сообщения в *json*-формате, где в каждом сообщении есть поле *type*. Эти сообщения обрабатываются и в зависимости от их типа вызываются определенные методы.

Голосование начинается, когда сервер высылает сообщение с типом *start\_voting*. В этом же сообщении содержится список «живых» игроков, которые являются кандидатами в голосовании. Алгоритм системы голосования представлен на рисунке 2.11.

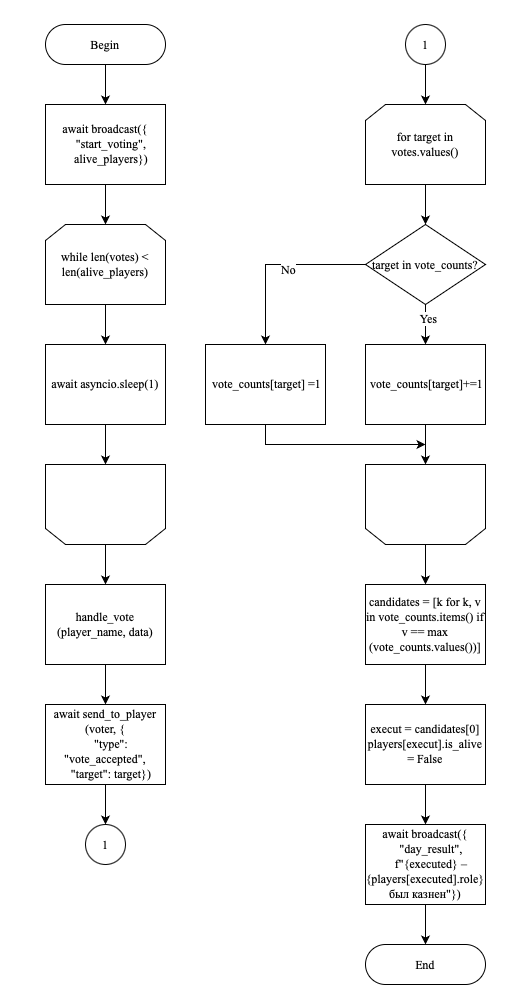


Рисунок 2.11 – Блок-схема алгоритма системы голосования

Каждый игрок выбирает нужного кандидата, которому хочет отдать голос. Далее программа подсчитывает, у кого больше голосов, и отправляет результат голосования всем игрокам. Если же одинаковое количество голосов наберут больше одного игрока, то никто не выбывает.

Обработка действий мафии и доктора происходит в течение ночной фазы. Алгоритм обработки ночных действий представлен на блок-схеме на рисунке 2.12.

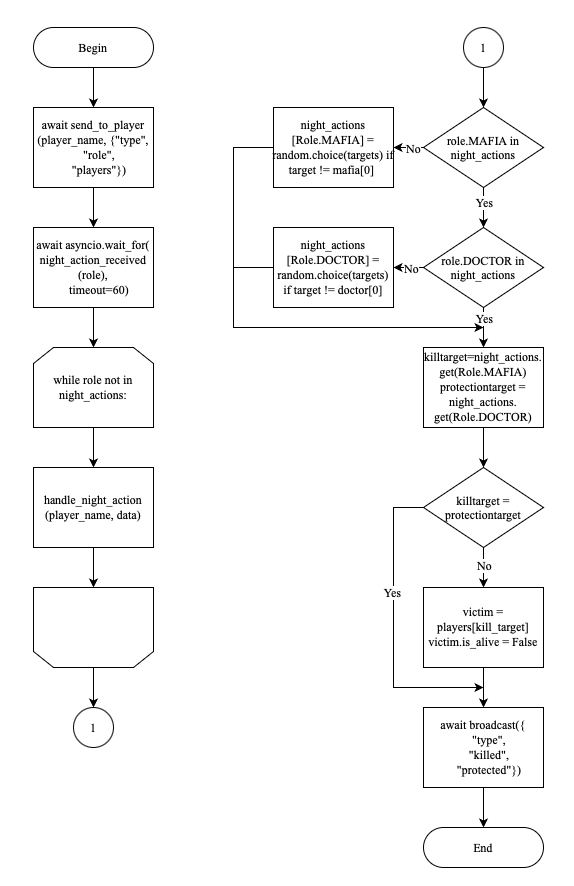


Рисунок 2.12 – Блок-схема алгоритма обработки ночных действий

С началом ночи игрокам со специальными ролями высвечивается окно ночной фазы с возможностью выбрать игрока, по отношению к которому будет совершено определенное действие. Выбор игроков отправляется на сервер, который проверяет, одинаковый ли выбор доктора или мафии, не пустой ли выбор, и отправляет игрокам результаты ночи.

# **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## **Взаимодействие клиентов с сервером**

Взаимодействие клиентов с сервером – это основополагающая часть игрового процесса. Посредством передачи сообщений в формате *JSON* сервер управляет игровыми фазами, обрабатывает результаты ночных действий и голосования, регулирует подключение и отключение игроков, определяет победителей и синхронизирует весь процесс игры. Клиенты, в свою очередь, отправляют сообщения с выполненными действиями – выбор цели для ночной фазы или голосования, обмениваются текстовыми сообщениями в чате, а также получают от сервера обновления о состоянии игры.

Как на стороне клиента, так и на стороне сервера есть специальные методы, которые отправляют сообщения, и методы, которые принимают сообщения.

За отправку сообщение со стороны клиента отвечает метод *send()*, который использует для этого установленное при подключении игрока к серверу *WebSocket*-соединение. Метод принимает словарь *data*, содержащий информацию о действии клиента, далее преобразует словарь в строку *JSON* с помощью *json.dumps* и отправляет *JSON*-сообщение на сервер через *WebSocket*-соединение (*self.ws.send*) [9].

Асинхронный метод *listen()* в классе *NetworkClient* постоянно ожидает входящие сообщения от сервера. После получения сообщения оно передается в метод *handle\_message()*. Метод анализирует входящие сообщения от сервера и вызывает соответствующие обработчики для выполнения действий на клиентской стороне. Он принимает словарь *data*, представляющий *JSON*-сообщение, полученное от сервера, извлекает тип сообщения (*“type”*) из данных, и в зависимости от него вызывает соответствующий метод для обработки. Например, *handle\_role\_assignment()* для назначения роли, если поле *«type»* имеет значение *«role»*, *handle\_phase\_change()* для смены игровой фазы, если поле *«type»* имеет значение *«phase»*, или *handle\_players\_update()* для обработки изменения количества игроков в комнате, если поле *«type»* имеет значение *«players\_update»*. Почти все такие обработчики отвечают за смену интерфейса и вызывают методы класса *GameUI*.

На стороне метод класса *Game*, который отвечает за принятие сообщение от клиентов также называется *handle\_message()*. Принимает в качестве аргументов строку *player\_name* (имя отправителя) и словарь *data*, представляющий собой *JSON*-сообщение от клиента. Метод анализирует тип сообщения (*«type»*) и в зависимости от него вызывает соответствующий метод для выполнения действий:

* *handle\_night\_action()* для обработки ночного действия, если тип *«night\_action»*;
* *handle\_vote()* для обработки голосования, если тип *«vote»*;
* *broadcast()*, который отправляет сообщение в чат всем игрокам, если тип *«chat»*.

За отправку сообщение клиентам отвечают два метода – *broadcast()* и *send\_to\_player()*. Метод *broadcast()* отправляет сообщение всем подключенным клиентам. Он перебирает все активные *WebSocket*-соединения в *self.connections* и отправляет сообщение каждому подключенному клиенту. Метод используется для глобальных обновлений, таких как обновление списка игроков, изменение игровой фазы, отправка результатов ночи и голосований, объявление победителей и обработка завершения игры. Метод *send\_to\_player()* отправляет сообщение конкретному игроку. Находит в словаре *self.connections* *WebSocket*-соединение игрока по его имени и отправляет сообщение через это соединение. Используется для отправки индивидуальных сообщений при назначении роли и запросе ночного действия, если игрок активен.

## **Подключение к игре**

Подключение игрока к серверу — это первый этап взаимодействия клиента и сервера в игре, который включает в себя установление *WebSocket*-соединения, проверку уникальности имени игрока, добавление игрока в список участников и уведомление всех клиентов об обновлении списка игроков.

За подключение клиентов к серверу отвечает метод *connect()* в классе *NetworkClient*. Основная задача этого метода — установить *WebSocket*-соединение с сервером и передать имя игрока. Сервер должен находиться по адресу *«ws://localhost:8000/ws»*. Если сервер недоступен или возникает ошибка, соединение завершается с исключением, которое можно обработать для отображения ошибки пользователю. После успешного подключения клиент отправляет серверу сообщение с типом *«join»* и своим именем. Имя игрока передается в *JSON*-формате и используется сервером для его идентификации. Затем запускается метод *listen()*, который обрабатывает входящие сообщения от сервера. Если сервер недоступен или имя игрока уже занято, клиент получает сообщение об ошибке и может попытаться подключиться снова.

Со стороны сервера за подключение игроков отвечает метод *handle\_connection()*. Задача этого метода — обработать запрос на подключение нового игрока, проверить его имя, сохранить соединение и уведомить других игроков. Когда клиент устанавливает соединение, он отправляет сообщение с именем игрока. Сервер проверяет, доступно ли это имя, и, если имя уже занято, соединение закрывается с кодом 4001 и сообщением *«Name already exists»* для предотвращения дублирования имен игроков. Если имя уникально, сервер добавляет нового игрока в список игроков и сохраняет его *WebSocket*-соединение в список *self.connections*.

После успешного подключения игрока сервер отправляет обновленный список всех игроков всем клиентам в виде сообщения типа *«players\_update»* содержит массив имен всех подключенных игроков, позволяя клиентам обновить интерфейс. После успешного подключения сервер переходит в цикл ожидания сообщений от клиента с помощью метода *handle\_message()*. Когда количество игроков достигает минимального значения, сервер инициирует начало игры. Если игрок отключается, сервер вызывает метод *handle\_disconnect()*, который удаляет игрока из списка активных и уведомляет остальных об изменении количества игроков.

## **Ночная фаза**

Ночная фаза в игре играет ключевую роль, так как в этот период происходят важные действия, связанные с выбором целей игроками, особенно для тех, кто выполняет роли мафии или доктора. Управление этой фазой осуществляется сервером через метод *game\_loop()*, который последовательно вызывает методы *night\_phase()* и *day\_phase()*. Сервер уведомляет клиентов о смене фазы, используя метод *broadcast()*. Это сообщение содержит информацию о текущей фазе и списке активных игроков. Клиенты обрабатывают это уведомление в методе *handle\_phase\_change()*, а интерфейс игры адаптируется в зависимости от фазы.

Во время ночной фазы клиентский интерфейс предоставляет возможность игрокам с определенными ролями (мафия или доктор) выбрать цель для своих действий. Для мафии интерфейс вызывается с помощью метода *\_create\_mafia\_interface()*, где игрок выбирает цель из выпадающего списка. Этот выбор отправляется на сервер с помощью метода *send()*. Аналогично, для доктора вызывается интерфейс с помощью метода *\_create\_doctor\_interface()*, и доктор также отправляет информацию о своей цели через тот же механизм. Клиентская сторона отвечает за отображение интерфейса и получение выбора игрока.

Когда устанавливается текущая фаза на «ночь», вызывается метод *night\_phase()*, который обрабатывает действия игроков с особыми ролями. Используя asyncio, сервер отправляет запросы с помощью метода *send\_to\_player()* игрокам с такими ролями и ожидает их действий. Для каждой роли создается задача, которая выполняется параллельно с помощью *asyncio.gather()*. Это позволяет серверу обрабатывать действия игроков одновременно, что значительно повышает эффективность и отзывчивость системы.

Если игрок бездействует, система назначает действие автоматически, чтобы избежать задержек в процессе. Действия игроков собираются с помощью метода *wait\_for\_night\_action()*, который ожидает поступления выборов от всех игроков. Все действия сохраняются в словаре *self.night\_actions*, что позволяет серверу отслеживать, кто какую цель выбрал.

Кроме того, важно отметить, что выбывшие игроки (те, кто был убит в предыдущей ночной фазе) не могут участвовать в действиях, но они все равно получают уведомление о текущей фазе и могут видеть, что происходит. Если выбывший игрок подключается к игре, он может видеть, что его роль завершена, и интерфейс изменится с помощью метода *not\_alive\_night\_phase()*, который информирует о том, что игрок мертв и должен дождаться утра.

После того как все действия собраны, вызывается метод *process\_night\_actions()*, который обрабатывает результаты. Сервер обрабатывает все собранные действия в определенном порядке – сначала выполняется лечение, чтобы защитить цель от возможного убийства. Если мафия выбирает жертву, а доктор лечит ту же цель, то лечение отменяет убийство. Если выбранная мафией жертва не совпадает с целью доктора, жертва погибает, и параметр *is\_alive* игрока устанавливается в *False*. После обработки ночных действий сервер отправляет всем игрокам сообщение *«night\_result»*, содержащее имя убитого игрока, имя защищенного игрока и роль убитого игрока, если он погиб.

Важно отметить, что асинхронная реализация позволяет избежать блокировок и обеспечивать плавный переход между процессами. Все действия выполняются в фоновом режиме, что позволяет серверу обрабатывать другие запросы во время ожидания ответов от игроков. Это особенно важно в условиях многопользовательской игры, где каждая секунда на счету.

На заключительном этапе сервер проверяет условия победы в методе *check\_win\_conditions()*. Затем он отправляет результаты клиентам, используя метод *broadcast()*. Это сообщение информирует всех клиентов о том, кто был убит, кто был защищен и какую роль играли участники.

## **Дневная фаза**

Дневная фаза в игре – это время, когда игроки обсуждают события ночной фазы и принимают решение о голосовании, чтобы устранить потенциальную угрозу в виде мафии. Этот процесс управляется сервером и включает взаимодействие между клиентами и сервером, обсуждение игроков в чате, а также обработку голосов.

После ночной фазы проверяются условия победы в методе *check\_win\_conditions* и, если победитель не определён, фаза меняется на «день» и запускается метод *day\_phase()*. Когда сервер переходит к дневной фазе, он отправляет уведомление всем клиентам о смене фазы с помощью метода *broadcast()*. На стороне клиента уведомление обрабатывается в методе *handle\_phase\_change()* и, если текущая фаза – дневная, то отображается интерфейс голосования и чата. Также игрок видит список живых участников и результаты ночной фазы.

Сервер предоставляет игрокам время для обсуждения с использованием таймеров *asyncio.sleep*. Во время дневной фазы игроки могут переключатся между интерфейсом голосования и чата, чтобы отправлять сообщения в общий чат. Сообщения содержат имя игрока и текст сообщения отправляются на сервер с помощью метода *send()*. Сервер принимает сообщения от клиентов и рассылает их всем участникам через метод *broadcast()*. Принимая от сервера данные типа *«chat\_message»*, клиент добавляет новое сообщение в словарь *self.chat\_history*. С помощью метода *show\_chat()* можно перейти в чат, чтобы отследить процесс обсуждения или отправить сообщения игрокам.

Во время голосования игроки получают список кандидатов для голосования. Сервер принимает голоса и обрабатывает их с помощью метода *process\_votes()*. Этот метод собирает все голоса в словарь *self.votes* и определяет, кто будет казнен. Если один кандидат получает большинство голосов, он выбывает и его *is\_alive* устанавливается в *False*, а при равенстве голосов никто не исключается. После голосования сервер снова проверяет условия победы и отправляет результаты голосования игрокам в сообщении *«day\_result»*, содержа имя казненного игрока или *None*, если никто не выбывает. Если игра продолжается, цикл переходит к следующему дню, увеличивая счётчик дней (*self.day\_number* += 1) и возвращаясь к ночной фазе через цикл *game\_loop()*.

Выбывшие игроки, которые были убиты на предыдущей ночной фазе, также могут видеть происходящее в дневной фазе, но они не могут участвовать в голосовании. Их интерфейс обновляется с помощью метода *not\_alive\_day\_phase()*, который информирует, что они мертвы и могут лишь следить за игрой. Выбывшие игроки всё равно могут просматривать чат и видеть обсуждения, но не могут отправлять сообщения или голосовать.

## **Завершение игры**

Завершение игры инициируется, когда проверяются условия победы. Это происходит в методе *check\_win\_conditions()* на сервере после каждой ночной и дневной фазы. Сервер проверяет:

* если количество игроков с ролью мафия (*Role.MAFIA*) стало равно нулю, то побеждают мирные жители (*villagers*);
* если количество мафии больше или равно количеству оставшихся мирных, то побеждает мафия.

Когда определен победитель, вызывается метод *end\_game()*. Этот метод устанавливает флаг *self.game\_started* в *False*, чтобы остановить игровой цикл, далее очищает списки игроков *self.players* и *WebSocket*-соединений *self.connections*. Сервер отправляет всем игрокам сообщение с типом *“game\_over”*, которое содержит информацию о том, кто победил – мафия или мирные жители, и список всех игроков вместе с их ролями.

На стороне клиента, когда клиент получает сообщение типа *“game\_over”* через *WebSocket*, метод *handle\_game\_over()* сбрасывает состояние клиента, устанавливая *self.is\_alive* в *True* для подготовки к следующей игре и очищая список игроков. Затем вызывается метод *show\_game\_over()* интерфейса пользователя, который отображает сообщение о победителе и список всех игроков с их ролями. Также добавляется кнопка «Вернуться в лобби», при нажатии которой *WebSocket*-соединение перезапускается, и игрок попадает на стартовый экран.

После завершения игрок может выйти из лобби и разорвать соединение с сервером. Разрыв соединения может также произойти в результате каких-либо ошибок или проблем с сетью. Это обрабатывается в методе *handle\_disconnect()* на сервере. В нем последовательно удаляется игрок из словаря *self.players*, далее удаляется его веб-сокет из словаря *self.connections*.

Разрыв соединение может произойти в том числе и во время игры, поэтому сервер проверяет был ли игрок, покинувший игру, жив. Если игрок был мертв, то игра продолжится. В случае, когда отключился «живой» игрок, сервер проверит, меньше ли количество активных игроков *current\_alive* минимально необходимого *self.min\_players*. Если игроков меньше, сервер завершает игру с причиной «Отмена игры из-за недостатка игроков» с помощью метода *end\_game(«game\_cancelled»)* и рассылает сообщение *«game\_over»*. После этого всем остальным игрокам придется заново ждать подключения минимально необходимого количества игроков.

После удаления игрока сервер формирует новый список активных игроков и рассылает обновление всем оставшимся игрокам через сообщение типа *«players\_update»*. Это сообщение содержит обновлённый список игроков, чтобы клиенты могли обновить интерфейс.

На стороне клиента разрыв соединения фиксируется в методе *listen()*, который постоянно прослушивает сообщения от сервера. Если *WebSocket*-соединение закрывается, клиент вызывает метод *handle\_disconnect()* класса *NetworkClient*. Метод вызывает функцию *show\_connection\_error()*, которая меняет интерфейс клиента, показывая ему сообщение о разрыве соединения и позволяя подключится к серверу заново, и метод *close\_ws()*, который закрывает текущее *WebSocket*-соединение. Состояние клиента сбрасывается, *self.ws* устанавливается в *None*, а имя игрока *self.name* очищается.

Также разрыв соединения может быть инициирован ошибками соединения на стороне сервера, которые могут быть вызваны из-за проблем с сетью или системных ошибок. Если сервер отключается, все клиенты теряют соединение, фиксируют потерю соединения через *ConnectionClosed* и показывают соответствующее сообщение пользователям. Восстановить возможно только через повторный запуск сервера и переподключение всех клиентов.

# **ТЕСТИРОВАНИЕ**

В процессе сетевой игры «Мафия» могут возникать исключительные ситуации, которые необходимо обрабатывать для обеспечения корректности игрового процесса и сохранения комфортной атмосферы для игроков. Главная исключительная ситуация – это отключение игрока от лобби.

Если игрок отключается из лобби, пока набор минимального необходимого количества игроков еще идет, его *WebSocket*-соединение просто закроется без каких-либо дополнительных обработок. Если игрок выходит из игры, когда сама игра уже началась, то программа среагирует по-разному в зависимости от того, был ли «жив» игрок. Если игрок был «мертв» и находился в игре только в качестве наблюдателя, то он также может отключится от лобби без каких-либо последствий. Если же игрок был «жив», то игра прекратиться, и остальным пользователям будет предоставлена возможность подключиться к новой комнате. Интерфейс с возможностью переподключиться к новой комнате представлен на рисунке 4.1.

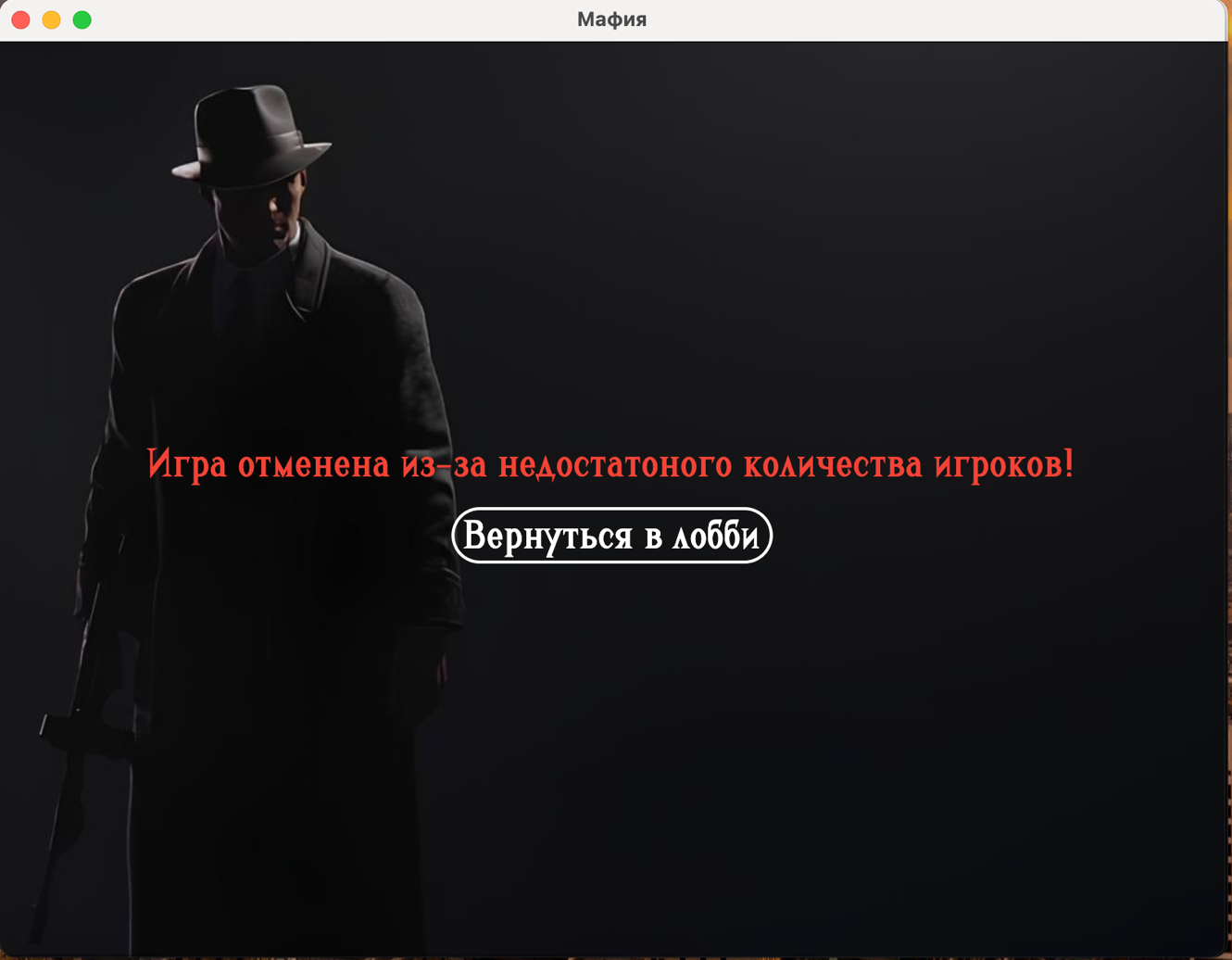


Рисунок 4.1 – Интерфейс с возможностью переподключиться к новой комнате

При нажатии на кнопку, пользователь снова попадет в меню, где сможет попробовать подключиться к новой комнате, просмотреть правила игры или выйти из приложения.

Еще одной важной исключительной ситуацией является ввод имени, которое уже используется в игре, при подключении к комнате. В таком случае программа попробует подключить пользователя к серверу, проверит в активных подключениях совпадение по имени и оповестит игрока о том, что такое имя уже занято. Интерфейс ошибки при попытке ввести уже активное имя представлен на рисунке 4.2.



Рисунок 4.2. – Интерфейс ошибки при попытке ввести уже активное имя

Еще одной исключительной ситуацией является попытка игрока подключиться к серверу, когда тот недоступен. Это может произойти в связи с проблемами с сетью или, если сервер не запущен. В таком случае программа также уведомит об этом пользователя и даст возможность вернуться в меню. Интерфейс ошибки при попытке подключиться к серверу, который недоступен, представлен на рисунке 4.3.

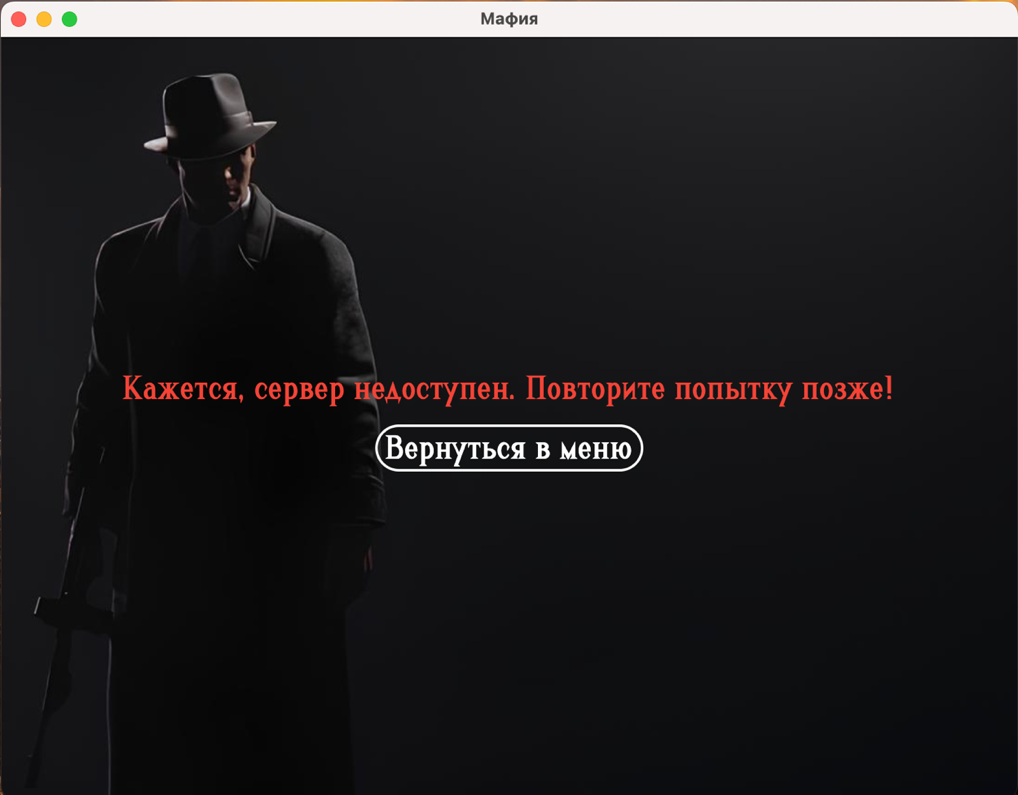


Рисунок 4.3 – Интерфейс ошибки при попытке подключиться к серверу, который недоступен

Также необходимо обработать ситуацию, когда сервер внезапно отключается во время игры, например из-за проблем с сетью. В таком случае необходимо оповестить клиентов, завершить игру и дать возможность игрокам перезапустить игру. Интерфейс в случае внезапного отключения сервера с возможностью выхода в меню представлен на рисунке 4.4.

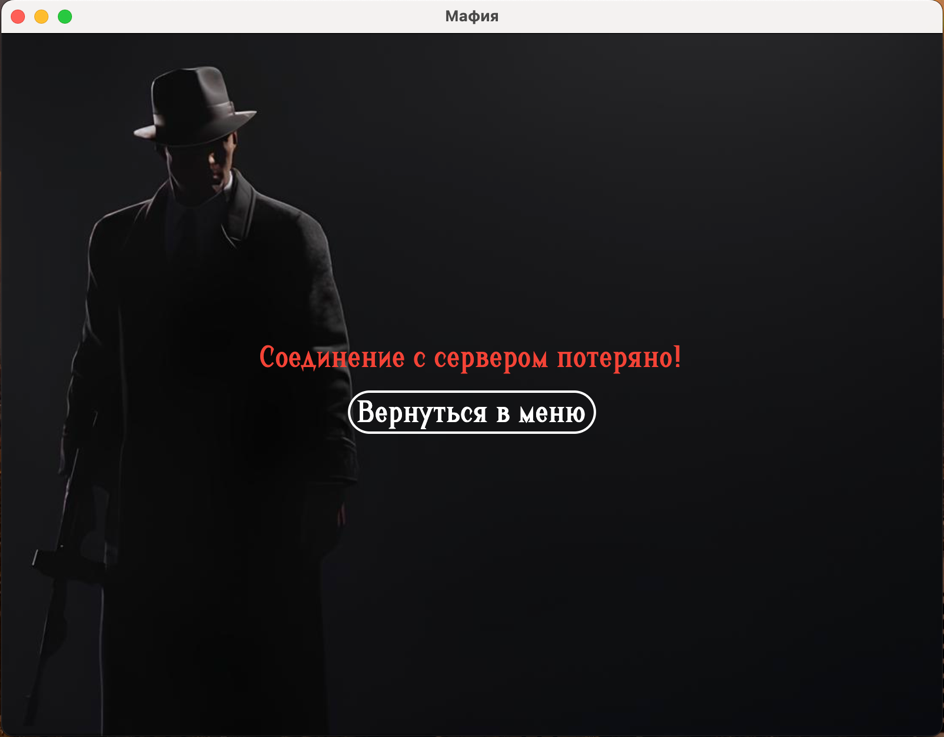


Рисунок 4.4 – Интерфейс в случае внезапного отключения сервера

# **ПРИМЕНЕНИЕ ПРОГРАММЫ**

Руководство пользователя предназначено для помощи пользователям в освоении интерфейса игры «Мафия», понимании элементов управления и выполнении основных игровых функций, таких как подключение к лобби, голосование, использование способностей ролей, и других действий.

## **Руководство пользователя**

Игровой интерфейс состоит из нескольких основных элементов, которые обеспечивают взаимодействие с игрой. На главном экране игроки видят меню игры, которое позволяет подключиться к комнате, посмотреть правила игры и выйти из приложения. Интерфейс меню представлен на рисунке 5.1.

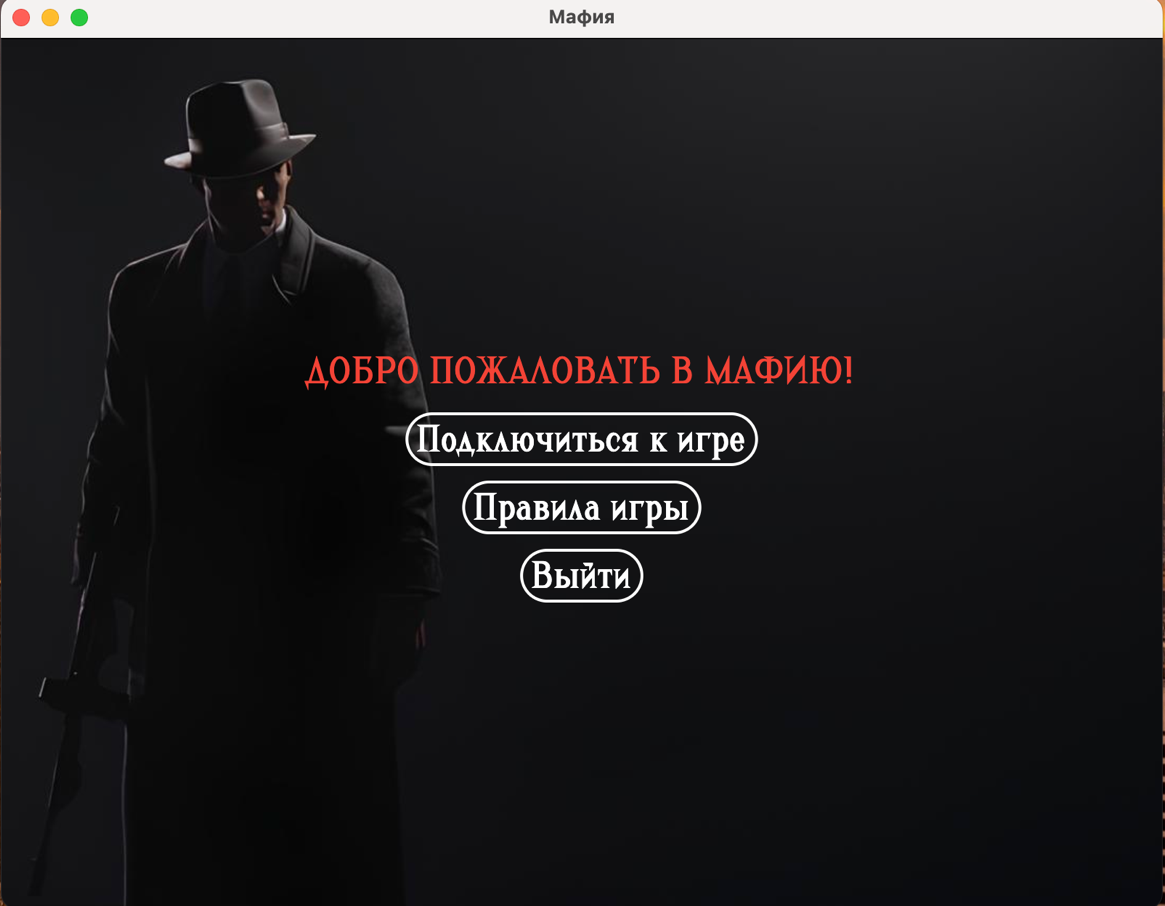


Рисунок 5.1. – Интерфейс меню игры

При нажатии на кнопку «Правила игры» игрок попадет на окно с правилами игры. В левом верхнем углу данного окна находится кнопка «Назад», нажав на которую, игрок вернется назад в меню. При нажатии на кнопку «Выйти» игрок выйдет из приложения. Интерфейс окна с правилами игры представлен на рисунке 5.2.

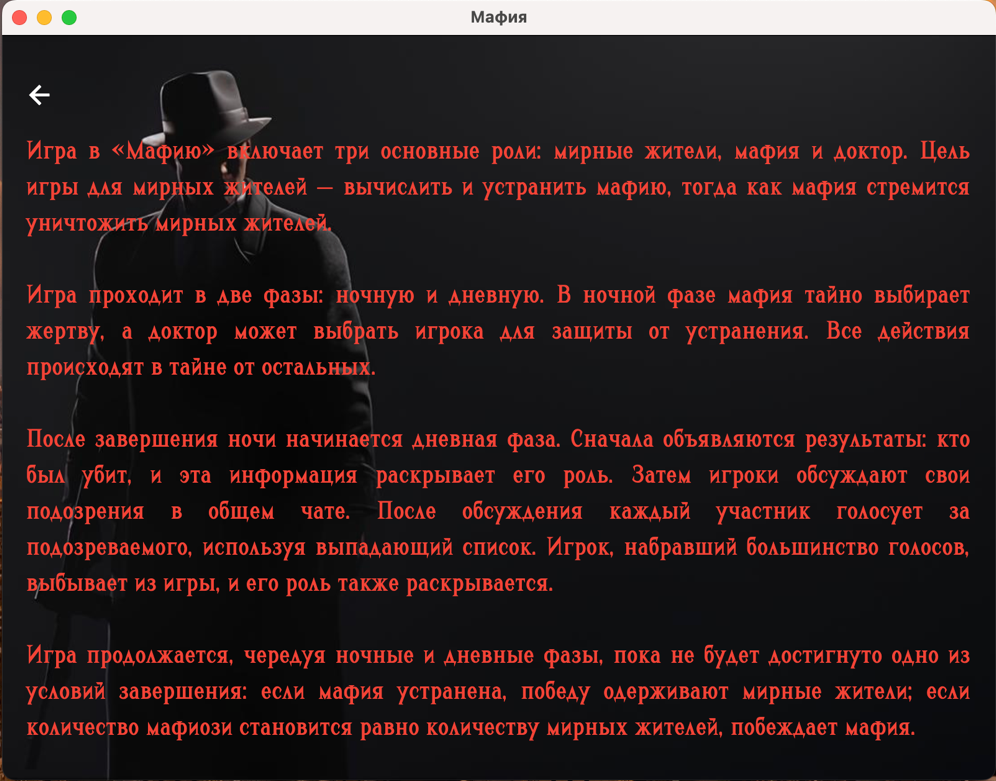


Рисунок 5.2.– Интерфейс окна с правилами игры

При нажатии на кнопку «Подключиться к игре» игрок попадет на окно подключения. На данном окне отображается поле для ввода уникального имени и кнопку «Присоединиться» для подключения к игре. Пока игрок не нажал кнопку подключиться, он может вернуться в меню игры, нажав на кнопку-стрелку в левом верхнем углу экрана. Интерфейс окна подключения представлен на рисунке 5.3.

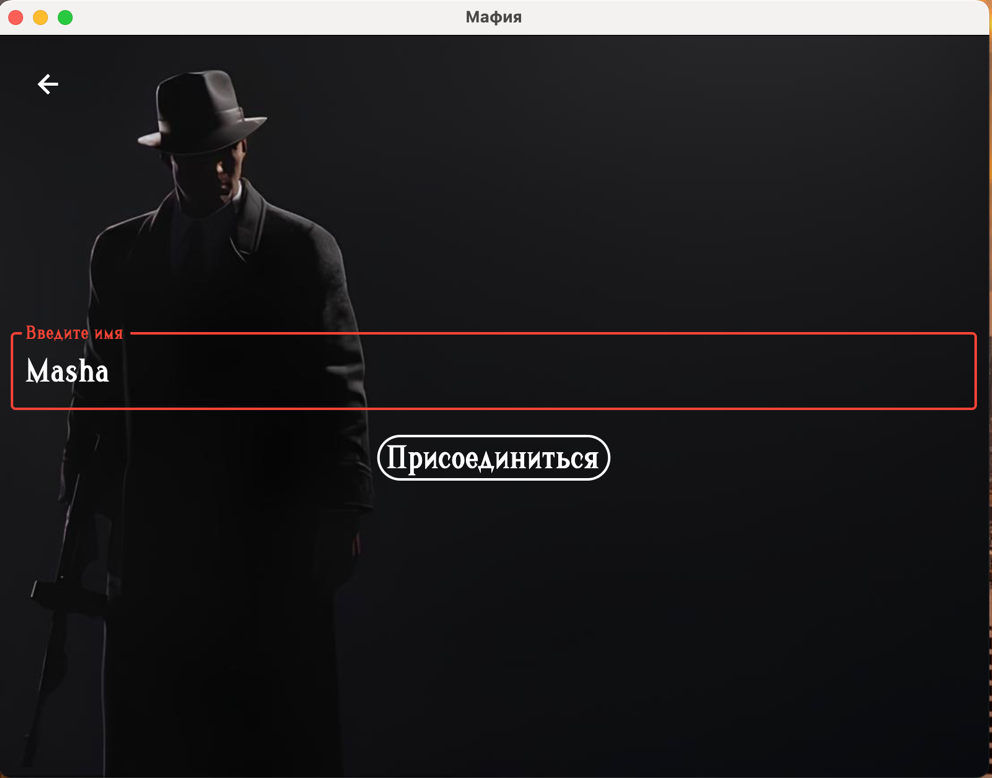


Рисунок 5.3 – Интерфейс главного окна

После успешного подключения игрок оказывается в игровом лобби. В лобби отображается список всех участников и индикатор прогресса, показывающий количество игроков, необходимых для начала игры. Когда минимальное количество игроков будет достигнуто, игра автоматически начнётся, и каждому игроку будет назначена роль.

На экране ролей игроки видят информацию о своей назначенной роли, список всех участников и обратный отсчет до начала игры. Игра проходит в две фазы: ночную и дневную. Во время ночной фазы, если игрок мафия, он выбирает цель для устранения из выпадающего списка и нажимает кнопку «Выбрать жертву». Если роль игрока – доктор, ему необходимо аналогично выбрать игрока для защиты и нажать кнопку «Защитить игрока». Если же игрок мирный житель, ему нужно дождаться завершения ночной фазы. Интерфейс ночной фазы мафии представлен на рисунке 5.4.

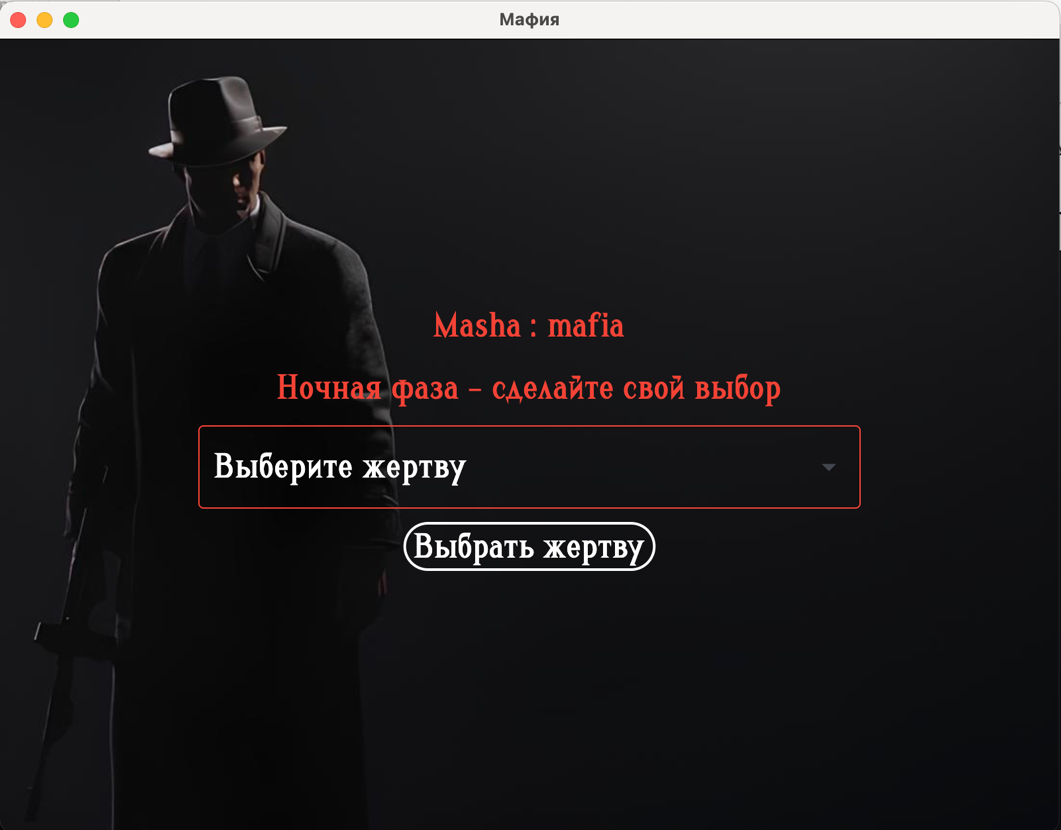


Рисунок 5.4 – Интерфейс ночной фазы

В дневной фазе игроки обсуждают подозреваемых в чате и голосуют за подозреваемого. На интерфейсе дневной фазы отображается выпадающий список с живыми игроками, кнопка «Проголосовать», чтобы отправить свой голос, и кнопка «Перейти в чат».

При переходе в чат отображается список всех сообщений от игроков, поле для ввода сообщений, кнопка «Отправить» и кнопка для перехода обратно на страницу с голосованием. Интерфейс чата представлен на рисунке 5.5.

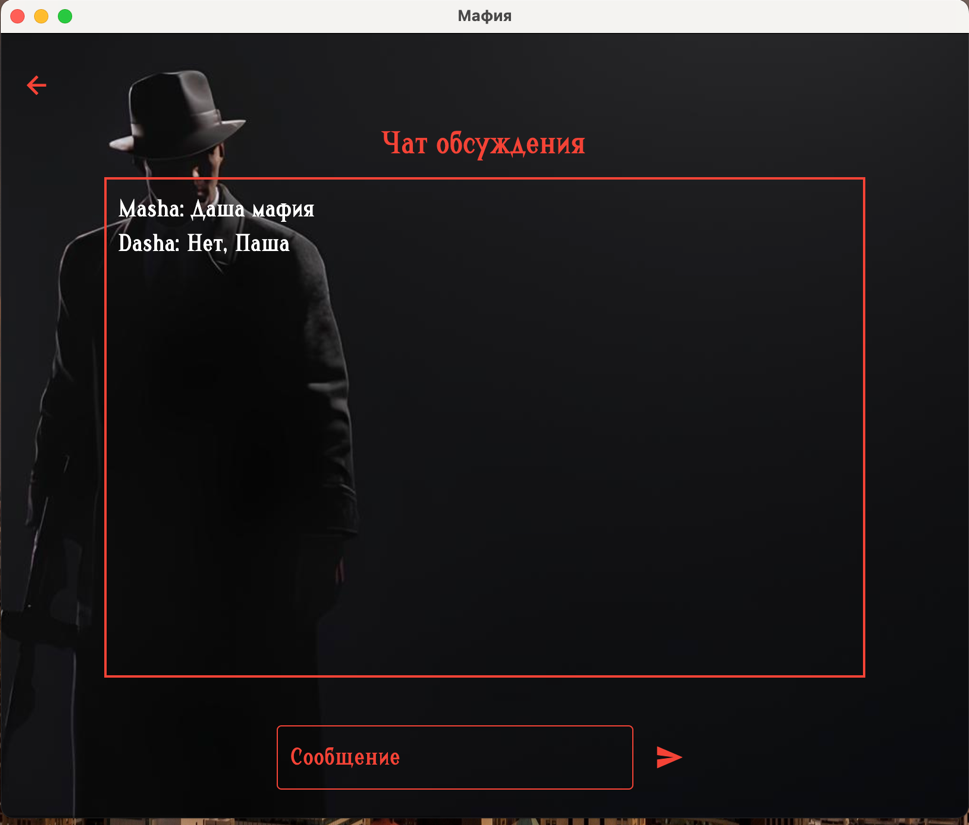


Рисунок 5.5 – Интерфейс чата

На интерфейсе завершения игры будет отображён победитель (мафия или мирные жители), список ролей всех игроков и кнопка «Вернуться в лобби». При нажатии на кнопку можно будет выйти из текущей игры и зайти в новую комнату. Интерфейс завершения игры отображен на рисунке 5.6.

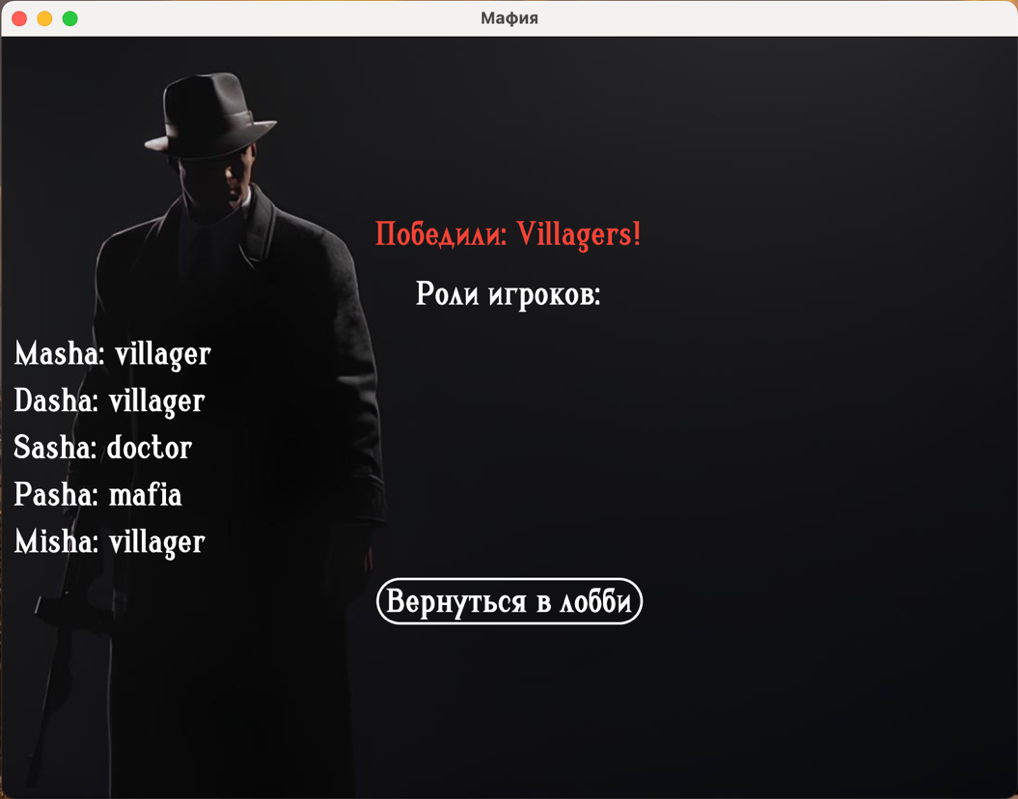


Рисунок 5.6 – Интерфейс завершения игры

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения данной курсовой работы была разработана полноценная сетевая игра «Мафия» с использованием современных технологий и инструментов, таких как *FastAPI* для серверной части, *Flet* для клиентской части и *WebSockets* для обеспечения двусторонней связи между игроками и сервером в реальном времени.

В рамках разработанной программы игроки могут подключаться к серверу, участвовать в игровых фазах, выполнять действия, характерные для их ролей, а также общаться через встроенный чат. Разработанный интерфейс обеспечивает взаимодействие с игрой и помогает пользователям погрузиться в процесс.

Одними из основных преимуществ являются использование *WebSocket*-соединения, которое позволяет обеспечить мгновенный обмен данными между игроками и сервером, и библиотеки *asyncio*, которая позволяет выполнять множество операций одновременно, не блокируя основной поток. К преимуществам также можно отнести современный и интуитивно понятный интерфейс, реализованный с помощью библиотеки *Flet*, и модульную структуру кода, которая делает код более читаемым и упрощает расширение функционала.

Среди недостатков текущей версии игры можно выделить отсутствие системы хранения данных пользователей. Игроки подключаются, указывая только имя, которое не сохраняется, что исключает возможность персонализации, создания аккаунтов и сохранения статистики. Еще одним ограничением является поддержка только одной игровой комнаты. Сервер позволяет вести лишь одну игру одновременно, что неудобно при большом числе игроков, желающих играть в разных группах. Чат в игре также реализован в базовом виде — он не поддерживает отправку мультимедиа, реакции или фильтрацию сообщений.

Анализ аналогов таких, как «*Polemica Game*» и «Мафия Онлайн», позволил выявить их ключевые особенности. Первая предлагает разнообразие игровых режимов и акцент на киберспортивные турниры, а вторая фокусируется на доступности и простоте для новичков. Разработанная игра «Мафия» имеет преимущества, включая *WebSocket*-соединение и модульную структуру кода, однако для конкуренции с аналогами необходимо добавить регистрацию, авторизацию и улучшить функционал чата.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Стандарт предприятия. Общие требования. СТП 01–2024 / Доманов А. Т., Сорока Н. И. – Минск : БГУИР, 2024. – 178 с.

[2] Документация языка программирования *Python* [Электронный рeсурс]. – Режим доступа: https://www.python.org

[3] Документация библиотеки *websocket* [Электронный рeсурс]. – Режим доступа: https://websockets.readthedocs.io/en/stable/

[4] Документация библиотеки *Flet* [Электронный рeсурс]. – Режим доступа: https://flet.dev

[5] Онлайн-платформа «*Polemica Game*» [Электронный рeсурс]. – Режим доступа: https://polemicagame.com

[6] Онлайн-платформа «Мафия Онлайн» [Электронный рeсурс]. – Режим доступа: https://www.mafiaonline.ru

[7] Объектно-ориентированное программирование на языке Python: Учебно-методическое пособие по дисциплине «Введение в компьютерные технологии»/ Задорожный С.С., Фадеев Е. П. – Физический факультет МГУ им. М. В. Ломоносова, 2022. – 40 с.

[8] Документация библиотеки *asyncio* [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/library/asyncio.html

[9] Документация библиотеки *json* [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/library/json.html

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

**client/main.py**

import flet as ft

from interface import GameUI

from network import NetworkClient

def main(page: ft.Page):

page.title = "Мафия"

page.horizontal\_alignment = "center"

page.vertical\_alignment = "center"

network = NetworkClient(page)

game\_ui = GameUI(page, network)

network.set\_game\_ui(game\_ui)

if \_\_name\_\_ == "\_\_main\_\_":

ft.app(target=main)

**client/interface.py**

import asyncio

import json

import flet as ft

from flet.core.types import MainAxisAlignment, TextAlign

from network import NetworkClient

w\_style = ft.TextStyle(font\_family="Ebbe", color=ft.colors.WHITE, size=26)

r\_style = ft.TextStyle(font\_family="Ebbe", color=ft.colors.RED, size=26)

l\_r\_style = ft.TextStyle(font\_family="Ebbe", color=ft.colors.RED, size=20)

l\_w\_style = ft.TextStyle(font\_family="Ebbe", color=ft.colors.WHITE, size=20)

with open('/Users/marialazarevic/PycharmProjects/mafia/client/role.json', 'r', encoding='utf-8') as file:

jsonroles = json.load(file)

class GameUI:

def \_\_init\_\_(self, page: ft.Page, network):

self.page = page

self.network = network

self.\_init\_ui()

self.show\_menu()

self.list\_players = []

self.role=""

self.name=""

def \_init\_ui(self):

self.current\_view = None

self.page.bgcolor = ft.colors.TRANSPARENT

self.show\_menu()

self.list\_players = []

self.role=""

self.name=""

def \_init\_ui(self):

self.current\_view = None

self.page.bgcolor = ft.colors.TRANSPARENT

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

self.player\_name = ft.TextField(label="Введите имя", text\_style=w\_style,focused\_border\_color=ft.colors.RED, cursor\_color=ft.colors.RED, label\_style=l\_r\_style)

self.connect\_btn = ft.OutlinedButton(content=ft.Text("Присоединиться", font\_family="Ebbe", size=26, color=ft.colors.WHITE), on\_click=self.connect,

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

self.connect\_btn.on\_hover = lambda e: self.handle\_hover(self.connect\_btn, e)

self.role\_display = ft.Text()

self.players\_list = ft.ListView(expand=True)

self.main\_column = ft.Column()

self.chat\_messages = ft.ListView(expand=True)

self.chat\_history = []

self.day\_phase\_controls = []

self.chat\_input = ft.TextField(

label="Сообщение",

multiline=True,

max\_lines=3,

text\_style=l\_w\_style,

border\_color=ft.colors.RED,

cursor\_color=ft.colors.RED,

focused\_border\_color=ft.colors.RED,

label\_style=l\_r\_style

)

self.chat\_btn = ft.IconButton(

icon=ft.icons.SEND,

icon\_color=ft.colors.RED,

on\_click=self.send\_chat\_message

)

@staticmethod

def handle\_hover(button, e):

if e.data == "true":

button.style.side = ft.BorderSide(2, ft.colors.RED)

button.content.color = ft.colors.RED

else:

button.style.side = ft.BorderSide(2, ft.colors.WHITE)

button.content.color = ft.colors.WHITE

button.update()

def show\_connect\_view(self):

self.clear\_page()

back\_button = ft.IconButton(

icon=ft.icons.ARROW\_BACK,

icon\_color=ft.colors.WHITE,

on\_click=lambda \_: self.show\_menu()

)

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(

src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER

)

)

form\_container = ft.Container(

content=ft.Column(

controls=[

self.player\_name,

self.connect\_btn

],

alignment=ft.MainAxisAlignment.CENTER,

horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

spacing=20

),

alignment=ft.alignment.center,

expand=True

)

back\_container = ft.Container(

content=back\_button,

alignment=ft.alignment.top\_left,

padding=10,

height=100

)

self.page.add(

ft.Stack(

controls=[

form\_container,

back\_container

],

expand=True

)

)

self.page.update()

async def connect(self, e):

self.name = self.player\_name.value

try:

connected = await asyncio.wait\_for(

self.network.connect(self.name),

timeout=3

)

if connected:

await asyncio.sleep(0.5)

if self.network.exception:

self.player\_name.value = ""

self.player\_name.label = "Это имя уже занято"

self.page.update()

self.network.exception = False

else:

self.update\_players\_list(self.network.players)

else:

self.show\_connection\_error()

except (asyncio.TimeoutError, ConnectionRefusedError):

self.show\_connection\_error()

except Exception as e:

print(f"Unexpected error: {e}")

self.show\_connection\_error()

def show\_role\_view(self, role: str, show\_duration):

self.clear\_page()

self.role=role

players\_list = ft.ListView(

controls=[

ft.Text(

p,

style=l\_w\_style,

text\_align=ft.TextAlign.CENTER # Центрирование текста

) for p in self.list\_players

],

height=200,

)

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

self.page.controls.append(

ft.Column(

controls=[

ft.Text(f"Ваша роль: {role}", style=r\_style),

ft.Divider(),

ft.Text("Состав игроков:", style=l\_w\_style),

ft.Container(

content=players\_list,

alignment=ft.alignment.center

),

ft.ProgressBar(width=300, color=ft.colors.RED),

ft.Text(f"Игра начнется через {show\_duration} секунд", style=w\_style)

],

alignment=ft.MainAxisAlignment.CENTER,

horizontal\_alignment=ft.CrossAxisAlignment.CENTER

)

)

self.page.update()

def show\_night\_phase(self, role: str, players: list):

self.clear\_page()

title=ft.Text(f"{self.name} : {role}", style=r\_style)

status = ft.Text("Ночная фаза - сделайте свой выбор", style=r\_style)

self.page.controls.append(title)

self.page.controls.append(status)

if role == "mafia":

self.\_create\_mafia\_interface(players)

elif role == "doctor":

self.\_create\_doctor\_interface(players)

else:

self.page.controls.append(

ft.Text("Вы спите... Дождитесь утра", style=w\_style)

)

self.page.update()

def not\_alive\_night\_phase(self, role):

self.clear\_page()

title = ft.Text(f"{self.name} : {role} – Вы мертвы", style=r\_style)

status = ft.Text("Ночная фаза. Дождитесь утра", style=r\_style)

self.page.controls.append(title)

self.page.controls.append(status)

self.page.update()

def \_create\_mafia\_interface(self, players):

self.target\_dropdown = ft.Dropdown(

options=[ft.DropdownOption(key=p, content=ft.Text(p, style=l\_w\_style)) for p in players],

text\_style=w\_style,

bgcolor=ft.colors.BLACK,

label="Выберите жертву",

border\_color=ft.colors.RED,

label\_style=w\_style,

width=500,

)

confirm\_btn = ft.OutlinedButton(

content=ft.Text("Выбрать жертву", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

confirm\_btn.on\_click= lambda e: self.\_send\_mafia\_choice(e,confirm\_btn)

confirm\_btn.on\_hover = lambda e: self.handle\_hover(confirm\_btn, e)

self.page.controls.extend([self.target\_dropdown, confirm\_btn])

def \_create\_doctor\_interface(self, players):

self.protect\_dropdown = ft.Dropdown(

options=[ft.dropdown.Option(key=p, content=ft.Text(p, style=l\_w\_style)) for p in players],

text\_style=w\_style,

bgcolor=ft.colors.BLACK,

label="Выберите игрока для защиты",

border\_color=ft.colors.RED,

label\_style=w\_style,

width=500,

)

confirm\_btn = ft.OutlinedButton(

content=ft.Text("Защитить игрока", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

confirm\_btn.on\_click = lambda e: self.\_send\_doctor\_choice(e, confirm\_btn)

confirm\_btn.on\_hover = lambda e: self.handle\_hover(confirm\_btn, e)

self.page.controls.extend([self.protect\_dropdown, confirm\_btn])

def \_send\_mafia\_choice(self, e, button):

if self.target\_dropdown.value:

self.page.run\_task(self.\_async\_send\_mafia\_choice, button)

async def \_async\_send\_mafia\_choice(self, button):

await self.network.send({

"type": "night\_action",

"target": self.target\_dropdown.value,

"role": "mafia"

})

self.page.controls.append(

ft.Text(f"Цель {self.target\_dropdown.value} выбрана...", style=w\_style)

)

print(f"Цель {self.target\_dropdown.value} выбрана...")

button.visible = False

button.disabled = True

self.page.update()

def \_send\_doctor\_choice(self, e, button):

if self.protect\_dropdown.value:

self.page.run\_task(self.\_async\_send\_doctor\_choice, button)

async def \_async\_send\_doctor\_choice(self, button):

await self.network.send({

"type": "night\_action",

"target": self.protect\_dropdown.value,

"role": "doctor"

})

self.page.controls.append(

ft.Text(f"Защита для {self.protect\_dropdown.value} отправлена...", style=w\_style)

)

print(f"Защита для {self.protect\_dropdown.value} отправлена...")

button.visible = False

button.disabled = True

self.page.update()

@staticmethod

def get\_day\_text(data):

killed=data.get("killed")

protected=data.get("protected")

role=data.get("role")

if killed==protected:

return f"Мафия пыталась убить {killed}. Но {protected} получил защиту от доктора."

else:

return f"Убит: {killed} ({role}). Получил защиту: {protected}"

def show\_day\_phase(self, players: list, night\_data):

self.current\_view = "day"

self.clear\_page()

title = ft.Text(f"{self.name} : {self.role}", style=r\_style)

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

self.vote\_dropdown=ft.Dropdown(

options=[ft.dropdown.Option(key=p, content=ft.Text(p, style=l\_w\_style)) for p in players],

border\_color=ft.colors.RED,

bgcolor=ft.colors.BLACK,

text\_style=w\_style,

label="Выберите игрока для голосования",

label\_style=w\_style,

width=500,

)

button=ft.OutlinedButton(

content=ft.Text("Проголосовать", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

))

chat\_button = ft.OutlinedButton(

content=ft.Text("Перейти в чат", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

))

chat\_button.on\_click = lambda e: self.show\_chat(e)

chat\_button.on\_hover = lambda e: self.handle\_hover(chat\_button, e)

button.on\_click = lambda e: self.send\_vote(e, button)

button.on\_hover=lambda e: self.handle\_hover(button, e)

text=self.get\_day\_text(night\_data)

self.page.controls.append(title)

self.page.controls.append(ft.Text("День. Обсуждение", style=w\_style))

self.page.controls.append(ft.Text(text, style=r\_style))

self.page.controls.append(ft.ListView(

controls=[ft.Text(p) for p in players],

height=200

))

self.page.controls.append(self.vote\_dropdown)

self.page.controls.append(button)

self.page.controls.append(chat\_button)

self.day\_phase\_controls = [

title,

ft.Text("День. Обсуждение", style=w\_style),

ft.Text(text, style=r\_style),

ft.ListView(controls=[ft.Text(p) for p in players], height=200),

self.vote\_dropdown,

button,

chat\_button

]

self.page.update()

def add\_chat\_message(self, sender: str, message: str):

self.chat\_history.append((sender, message))

if self.current\_view == "chat":

self.chat\_messages.controls = [

ft.Text(f"{s}: {m}", style=l\_w\_style)

for s, m in self.chat\_history

]

try:

self.chat\_messages.scroll\_to(offset=-1, duration=100)

self.page.update()

except Exception as e:

print(f"Ошибка обновления чата: {e}")

async def send\_chat\_message(self, e):

if self.chat\_input.value.strip():

await self.network.send({

"type": "chat",

"message": self.chat\_input.value

})

self.chat\_input.value = ""

self.page.update()

def not\_alive\_day\_phase(self, players, night\_data):

self.clear\_page()

self.chat\_messages.controls.clear()

title = ft.Text(f"{self.name} : {self.role} – Вы мертвы", style=r\_style)

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

text = self.get\_day\_text(night\_data)

self.page.controls.append(title)

self.page.controls.append(ft.Text("День. Обсуждение", style=w\_style))

self.page.controls.append(ft.Text(text, style=r\_style))

self.page.controls.append(ft.ListView(

controls=[ft.Text(p) for p in players],

height=200

))

self.page.update()

def show\_day\_result(self, data):

self.clear\_page()

self.chat\_messages.controls.clear()

title1 = ft.Text(f"{self.name} : {self.role}", style=r\_style)

title = ft.Text(data["message"], style=r\_style)

text=ft.Text("Ожидание ночной фазы...", style=w\_style)

self.page.controls.append(ft.Text("День. Обсуждение", style=w\_style))

self.page.controls.append(title1)

self.page.controls.append(title)

self.page.controls.append(text)

self.page.controls.append(ft.ProgressBar(color=ft.colors.RED, width=300))

self.page.update()

def show\_chat(self, e):

self.clear\_page()

self.current\_view = "chat"

self.chat\_messages.controls = [

ft.Text(f"{s}: {m}", style=l\_w\_style)

for s, m in self.chat\_history

]

back\_button = ft.IconButton(

icon=ft.icons.ARROW\_BACK,

icon\_color=ft.colors.RED,

on\_click=lambda \_: self.show\_day\_interface()

)

chat\_container = ft.Container(

content=self.chat\_messages,

width=self.page.width \* 0.8,

height=self.page.height \* 0.6,

border=ft.border.all(2, ft.colors.RED),

padding=10

)

input\_row = ft.Row(

[

self.chat\_input,

self.chat\_btn

],

alignment=ft.MainAxisAlignment.CENTER,

width=self.page.width \* 0.8

)

self.page.add(

ft.Column(

[

ft.Row([back\_button], alignment=ft.MainAxisAlignment.START),

ft.Text("Чат обсуждения", style=r\_style),

chat\_container,

ft.Divider(height=20, color=ft.colors.TRANSPARENT),

input\_row

],

horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

scroll=ft.ScrollMode.ALWAYS

)

)

self.page.update()

def show\_day\_interface(self):

self.clear\_page()

self.current\_view = "day"

self.page.controls.extend(self.day\_phase\_controls)

self.page.update()

def send\_vote(self, e, button):

if self.vote\_dropdown.value:

self.page.run\_task(self.\_async\_send\_vote, button)

async def \_async\_send\_vote(self, button):

await self.network.send({

"type": "vote",

"target": self.vote\_dropdown.value,

"player": self.name

})

self.page.controls.append(

ft.Text(f"Голос отправлен...", style=w\_style)

)

print(f"Голос {self.vote\_dropdown.value} отправлен")

button.visible = False

button.disabled = True

self.page.update()

def show\_countdown(self, seconds: int, players: list):

self.list\_players=players

players\_list = ft.ListView(

controls=[

ft.Text(

p,

style=l\_w\_style,

text\_align=ft.TextAlign.CENTER

) for p in self.list\_players

]

)

self.clear\_page()

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

self.page.add(

ft.Column(

[

ft.Row(

[ft.Text(f"Старт через: {seconds} сек", style=r\_style)],

alignment=ft.MainAxisAlignment.CENTER),

ft.Container(

content=players\_list,

alignment=ft.alignment.center

),

ft.Row(

[ft.ProgressBar(color=ft.colors.RED)],

alignment=ft.MainAxisAlignment.CENTER)

],

alignment=ft.MainAxisAlignment.CENTER,

horizontal\_alignment=ft.CrossAxisAlignment.CENTER

)

)

self.page.update()

def clear\_page(self):

self.page.controls.clear()

self.page.update()

def update\_players\_list(self, players: list):

self.clear\_page()

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

players\_list = ft.ListView(

controls=[

ft.Text(

p,

style=r\_style,

text\_align=ft.TextAlign.CENTER

) for p in players

]

)

progress = ft.ProgressBar(

color=ft.colors.RED,

)

self.page.controls.append(

ft.Column(

controls=[

ft.Row(

[ft.Text("Ожидание начала игры...", style=w\_style)],

alignment=ft.MainAxisAlignment.CENTER

),

ft.Container(progress, padding=10),

ft.Row(

[ft.Text(f"{len(players)}/5 игроков", style=r\_style)],

alignment=ft.MainAxisAlignment.CENTER

),

ft.Container(

content=players\_list,

alignment=ft.alignment.center

),

],

alignment=ft.MainAxisAlignment.CENTER,

horizontal\_alignment=ft.CrossAxisAlignment.CENTER

)

)

self.page.update()

def show\_game\_over(self, winner: str, roles: dict):

self.clear\_page()

self.current\_view = "game\_over"

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

role\_list = ft.ListView(expand=True)

for name, role in roles.items():

role\_list.controls.append(

ft.Text(f"{name}: {role}", style=w\_style)

)

button = ft.OutlinedButton(

content=ft.Text("Вернуться в лобби", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

on\_click=lambda e: self.page.run\_task(self.\_return\_to\_lobby, e),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

button.on\_hover=lambda e: self.handle\_hover(button, e)

self.page.add(

ft.Column(

[

ft.Text(f"Победили: {winner.capitalize()}!", style=r\_style),

ft.Text("Роли игроков:", style=w\_style),

ft.Container(role\_list, alignment=ft.alignment.center),

button

],

alignment=ft.MainAxisAlignment.CENTER,

horizontal\_alignment=ft.CrossAxisAlignment.CENTER

)

)

self.page.update()

async def \_return\_to\_lobby(self, e):

await self.network.close\_ws()

self.show\_menu()

def show\_game\_cancelled(self):

self.clear\_page()

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

button = ft.OutlinedButton(

content=ft.Text("Вернуться в лобби", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

on\_click=lambda e: self.page.run\_task(self.\_return\_to\_lobby, e), # Исправлено здесь

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

button.on\_hover = lambda e: self.handle\_hover(button, e)

self.page.add(

ft.Column(

[

ft.Text("Игра отменена из-за недостатоного количества игроков!", style=r\_style),

button

],

alignment=ft.MainAxisAlignment.CENTER,

horizontal\_alignment=ft.CrossAxisAlignment.CENTER

)

)

self.page.update()

def show\_menu(self):

self.clear\_page()

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

title=ft.Text("ДОБРО ПОЖАЛОВАТЬ В МАФИЮ!", style=r\_style)

connect\_button = ft.OutlinedButton(

content=ft.Text("Подключиться к игре", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

on\_click = lambda e: self.show\_connect\_view(),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

connect\_button.on\_hover = lambda e: self.handle\_hover(connect\_button, e)

rules\_button = ft.OutlinedButton(

content=ft.Text("Правила игры", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

on\_click=lambda e: self.show\_rules(),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

rules\_button.on\_hover = lambda e: self.handle\_hover(rules\_button, e)

exit\_button = ft.OutlinedButton(

content=ft.Text("Выйти", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

on\_click=lambda e: self.close\_app(),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

exit\_button.on\_hover = lambda e: self.handle\_hover(exit\_button, e)

self.page.controls.append(title)

self.page.controls.append(connect\_button)

self.page.controls.append(rules\_button)

self.page.controls.append(exit\_button)

self.page.update()

def show\_rules(self):

self.clear\_page()

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER)

)

back\_button = ft.IconButton(

icon=ft.icons.ARROW\_BACK,

icon\_color=ft.colors.WHITE,

on\_click=lambda \_: self.show\_menu()

)

rules\_text = ft.Text(jsonroles['rules'], style=l\_r\_style, text\_align=TextAlign.JUSTIFY)

rules=ft.Container(

content=rules\_text,

width=0.95 \* self.page.window.width,

alignment=ft.alignment.center,

expand=True)

self.page.add(

ft.Column(

[

ft.Row([back\_button], alignment=ft.MainAxisAlignment.START),

rules

],

horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

scroll=ft.ScrollMode.ALWAYS

)

)

self.page.update()

def close\_app(self, event=None):

if self.page:

self.page.window.close()

self.page.update()

def show\_connection\_error(self):

self.clear\_page()

self.page.decoration = ft.BoxDecoration(

image=ft.DecorationImage(

src="/Users/marialazarevic/Downloads/MAFIA DE Wallpapers black suit-2.jpg",

fit=ft.ImageFit.COVER

)

)

error\_text = ft.Text("Кажется, сервер недоступен. Повторите попытку позже!", style=r\_style)

back\_button = ft.OutlinedButton(

content=ft.Text("Вернуться в меню", font\_family="Ebbe", size=26, color=ft.colors.WHITE),

on\_click=lambda e: self.show\_menu(),

style=ft.ButtonStyle(

side=ft.BorderSide(2, ft.colors.WHITE)

)

)

back\_button.on\_hover = lambda e: self.handle\_hover(back\_button, e)

self.page.add(

ft.Column(

[

error\_text,

back\_button

],

alignment=ft.MainAxisAlignment.CENTER,

horizontal\_alignment=ft.CrossAxisAlignment.CENTER

)

)

self.page.update()

**client/network.py**

import websockets

import json

import asyncio

from typing import Optional, Dict, Any

class NetworkClient:

def \_\_init\_\_(self, page):

self.page = page

self.ws: Optional[websockets.client.WebSocketClientProtocol] = None

self.name = ""

self.role: Optional[str] = None

self.players: Dict[str, Any] = {}

self.game\_ui = None

self.night\_data: Dict[str, Any] = {}

self.is\_alive = True

self.exception=False

async def connect(self, name: str) -> bool:

try:

self.name = name

self.ws = await websockets.connect(

"ws://localhost:8000/ws",

open\_timeout=3

)

await self.ws.send(json.dumps({"type": "join", "name": name}))

asyncio.create\_task(self.listen())

return True

except (ConnectionRefusedError, OSError, asyncio.TimeoutError):

print("Сервер недоступен")

return False

except Exception as e:

print(f"Connection error: {e}")

return False

async def listen(self):

try:

async for message in self.ws:

data = json.loads(message)

await self.handle\_message(data)

except websockets.exceptions.ConnectionClosedOK:

print("Соединение закрыто нормально")

await self.handle\_disconnect()

except websockets.exceptions.ConnectionClosedError as e:

print(f"Соединение закрыто с ошибкой: {e}")

await self.handle\_disconnect()

except Exception as e:

print(f"Неизвестная ошибка: {e}")

await self.handle\_disconnect()

async def send(self, data: dict):

await self.ws.send(json.dumps(data))

print("Действие отправлено")

async def handle\_message(self, data: dict):

message\_type = data["type"]

print(f"Получено сообщение типа: {message\_type}")

if message\_type == "show\_roles":

if self.game\_ui:

self.game\_ui.show\_role\_view(self.role, data["duration"])

elif message\_type == "role":

await self.handle\_role\_assignment(data)

elif message\_type=="night\_result":

self.night\_data=data

if data.get("killed") == self.name and data.get("killed") != data.get("protected"):

self.is\_alive = False

elif message\_type == "day\_result":

if data.get("executed") == self.name:

self.is\_alive = False

self.game\_ui.show\_day\_result(data)

elif message\_type == "phase":

print(data["phase"])

await self.handle\_phase\_change(data)

elif message\_type == "players\_update":

await self.handle\_players\_update(data)

elif message\_type == "game\_over":

if data["winner"] == "game\_cancelled":

await self.handle\_game\_cancelled(data)

else: await self.handle\_game\_over(data)

elif message\_type == "error":

await self.handle\_error(data)

elif message\_type == "game\_starting":

if self.game\_ui:

print(f"Игра начнется через {data['seconds']} секунд")

self.game\_ui.show\_countdown(data["seconds"], data["players"])

elif message\_type == "game\_cancelled":

if self.game\_ui:

print("Игра отменена: недостаточно игроков")

elif message\_type == "chat\_message":

if self.game\_ui:

self.game\_ui.add\_chat\_message(data["sender"], data["message"])

async def handle\_role\_assignment(self, data: dict):

self.role = data["role"]

async def handle\_phase\_change(self, data: dict):

phase = data["phase"]

players=data.get("players", [])

for player in players:

print(player)

if self.game\_ui:

if phase == "night":

if self.is\_alive:

self.game\_ui.show\_night\_phase(

self.role,

data.get("players", [])

)

else: self.game\_ui.not\_alive\_night\_phase(self.role)

elif phase == "day":

if self.is\_alive:

self.game\_ui.show\_day\_phase(

data.get("players", []),

self.night\_data

)

else: self.game\_ui.not\_alive\_day\_phase(data.get("players", []), self.night\_data)

async def handle\_players\_update(self, data: dict):

self.players = data["players"]

players = data.get("players", [])

for player in players:

print(player)

if self.game\_ui:

self.game\_ui.update\_players\_list(self.players)

async def handle\_game\_over(self, data: dict):

self.is\_alive = True

self.role = None

self.players = {}

if self.game\_ui:

self.game\_ui.show\_game\_over(data["winner"], data["roles"])

await self.close\_ws()

async def handle\_error(self, data: dict):

error\_msg = data["message"]

if self.game\_ui:

self.game\_ui.show\_error(error\_msg)

async def handle\_disconnect(self):

await self.close\_ws()

async def close\_ws(self):

if self.ws:

try:

await self.ws.close()

except Exception as e:

print(f"Ошибка при закрытии соединения: {e}")

finally:

self.ws = None

self.name = ""

def set\_game\_ui(self, game\_ui):

self.game\_ui = game\_ui

async def handle\_game\_cancelled(self, data):

self.is\_alive = True

self.role = None

self.players = {}

if self.game\_ui:

self.game\_ui.show\_game\_cancelled()

await self.close\_ws()

**server/game.py**

from typing import Dict, List, Optional  
from player import Player, Role  
from fastapi import WebSocket  
import random  
import json  
import asyncio  
  
class Game:  
 def \_\_init\_\_(self):  
 self.players: Dict[str, Player] = {}  
 self.connections: Dict[str, WebSocket] = {}  
 self.game\_started = False  
 self.phase = "lobby"  
 self.day\_number = 1  
 self.night\_actions: Dict[Role, str] = {}  
 self.votes: Dict[str, str] = {}  
 self.min\_players = 3  
 self.start\_task = None  
 self.lock = asyncio.Lock()  
  
 async def handle\_connection(self, websocket: WebSocket, player\_name: str):  
 if player\_name in self.players:  
 await websocket.close(code=4001, reason="Name already exists")  
 return  
 self.connections[player\_name] = websocket  
 self.players[player\_name] = Player(name=player\_name)  
 await self.broadcast({  
 "type": "players\_update",  
 "players": [p.name for p in self.players.values()]  
 })  
 print(f"Новый игрок: {player\_name}, Всего игроков: {len(self.players)}")  
 async with self.lock:  
 if (  
 len(self.players) >= self.min\_players  
 and not self.game\_started  
 and (not self.start\_task or self.start\_task.done())  
 ):  
 self.start\_task = asyncio.create\_task(self.start\_game())  
 print("Игра скоро начнется")  
 try:  
 while True:  
 data = await websocket.receive\_json()  
 await self.handle\_message(player\_name, data)  
 except Exception as e:  
 print(f"Error with {player\_name}: {e}")  
 finally:  
 await self.handle\_disconnect(player\_name)  
  
 async def handle\_message(self, player\_name: str, data: dict):  
 if data["type"] == "night\_action":  
 print(f"Получено сообщение {data["type"]}")  
 print(data["role"])  
 print(data["target"])  
 await self.handle\_night\_action(player\_name, data)  
 elif data["type"] == "vote":  
 print(f"Получено сообщение {data["type"]}")  
 await self.handle\_vote(player\_name, data)  
 elif data["type"] == "chat":  
 await self.broadcast({  
 "type": "chat\_message",  
 "sender": player\_name,  
 "message": data["message"],  
 "timestamp": asyncio.get\_event\_loop().time()  
 })  
  
 async def handle\_disconnect(self, player\_name: str):  
 print(f"Игрок {player\_name} отключился")  
 was\_alive = False  
 if player\_name in self.players:  
 was\_alive = self.players[player\_name].is\_alive  
 del self.players[player\_name]  
 if player\_name in self.connections:  
 try:  
 await self.connections[player\_name].close()  
 except Exception as e:  
 print(f"Ошибка при закрытии соединения: {e}")  
 finally:  
 del self.connections[player\_name]  
  
 if self.game\_started:  
 if was\_alive:  
 alive\_players = [p.name for p in self.players.values() if p.is\_alive]  
 await self.broadcast({  
 "type": "players\_update",  
 "players": alive\_players  
 })  
 current\_alive = len([p for p in self.players.values() if p.is\_alive])  
 if current\_alive < self.min\_players:  
 await self.end\_game("game\_cancelled")  
 else:  
 # Обновляем список игроков для всех, если игра еще не началась  
 await self.broadcast({  
 "type": "players\_update",  
 "players": [p.name for p in self.players.values()]  
 })  
  
 async def start\_game(self):  
 try:  
 async with self.lock:  
 if self.game\_started:  
 return  
 self.game\_started = True  
 initial\_players = list(self.players.keys())  
  
 for i in range(5, 0, -1):  
 await self.broadcast({  
 "type": "game\_starting",  
 "seconds": i,  
 "players": initial\_players  
 })  
 await asyncio.sleep(1)  
  
 current\_players = list(self.players.keys())  
 if (len(current\_players) < self.min\_players  
 or set(current\_players) != set(initial\_players)):  
 await self.broadcast({"type": "game\_cancelled"})  
 print("[Сервер] Игра отменена")  
 return  
 roles = [Role.MAFIA, Role.DOCTOR] + [Role.VILLAGER] \* (len(initial\_players) - 2)  
 random.shuffle(roles)  
 for player in self.players.values():  
 player.role = roles.pop()  
 await self.send\_to\_player(player.name, {  
 "type": "role",  
 "role": player.role.value,  
 "players": initial\_players  
 })  
  
 # Новая задержка для показа ролей  
 for i in range(5, 0, -1):  
 await self.broadcast({"type": "show\_roles", "duration": i})  
 await asyncio.sleep(1)  
 #await asyncio.sleep(5)  
 # Проверка подключения после показа ролей  
 current\_players = list(self.players.keys())  
 if (len(current\_players) < self.min\_players  
 or set(current\_players) != set(initial\_players)):  
 await self.broadcast({"type": "game\_cancelled"})  
 print("[Сервер] Игра отменена после показа ролей")  
 return  
 # Запуск основного цикла  
 await self.game\_loop()  
 except Exception as e:  
 print(f"[Сервер] Ошибка: {str(e)}")  
 await self.broadcast({"type": "game\_cancelled"})  
 finally:  
 self.start\_task = None  
  
 async def game\_loop(self):  
 while self.game\_started:  
 self.phase = "night"  
 await self.night\_phase()  
 await asyncio.sleep(1)  
 await self.check\_win\_conditions()  
 if not self.game\_started:  
 break  
 self.phase = "day"  
 await self.day\_phase()  
 await self.check\_win\_conditions()  
 self.day\_number += 1  
 await asyncio.sleep(2)  
  
 async def night\_phase(self):  
 await self.broadcast({  
 "type": "phase",  
 "phase": "night",  
 "players": [p.name for p in self.players.values() if p.is\_alive]  
 })  
 await asyncio.sleep(2)  
 self.night\_actions.clear()  
 mafia\_players = [p for p in self.players.values() if p.role == Role.MAFIA]  
 doctor\_players = [p for p in self.players.values() if p.role == Role.DOCTOR]  
 tasks = []  
 for player in mafia\_players + doctor\_players:  
 tasks.append(  
 self.wait\_for\_night\_action(player.name, player.role)  
 )  
 await asyncio.gather(\*tasks)  
 await self.process\_night\_actions()  
  
 async def wait\_for\_night\_action(self, player\_name: str, role: Role):  
 try:  
 await self.send\_to\_player(player\_name, {  
 "type": "request\_night\_action",  
 "role": role.value,  
 "players": [p.name for p in self.players.values() if p.is\_alive]  
 })  
 await asyncio.wait\_for(  
 self.night\_action\_received(role),  
 timeout=60  
 )  
 except asyncio.TimeoutError:  
 print(f"Таймаут ночного действия для {role}")  
  
 async def night\_action\_received(self, role: Role):  
 while role not in self.night\_actions:  
 await asyncio.sleep(1)  
  
 async def handle\_night\_action(self, player\_name: str, data: dict):  
 role = self.players[player\_name].role  
 self.night\_actions[role] = data["target"]  
 #await self.check\_night\_actions\_complete()  
  
 async def check\_night\_actions\_complete(self):  
 pass  
  
 async def process\_night\_actions(self):  
 print("Обработка ночных действий")  
 kill\_target = self.night\_actions.get(Role.MAFIA)  
 protection\_target = self.night\_actions.get(Role.DOCTOR)  
 if kill\_target and kill\_target in self.players:  
 victim = self.players[kill\_target]  
 if kill\_target != protection\_target:  
 victim.is\_alive = False  
 role\_info = victim.role.value  
 else:  
 role\_info = None  
 kill\_target = None  
 result\_message = {  
 "type": "night\_result",  
 "killed": kill\_target,  
 "protected": protection\_target,  
 "role": role\_info  
 }  
 await self.broadcast(result\_message)  
 print(f"Отправлены результаты ночи: {result\_message}")  
 await asyncio.sleep(3)  
  
 async def day\_phase(self):  
 await self.broadcast({  
 "type": "phase",  
 "phase": "day",  
 "players": [p.name for p in self.players.values() if p.is\_alive]  
 })  
 for i in range(3, 0, -1):  
 await self.broadcast({"type": "day\_countdown", "message": f"Рассвет через {i}..."})  
 await asyncio.sleep(1)  
 self.votes.clear()  
 alive\_players = [p.name for p in self.players.values() if p.is\_alive]  
 await self.broadcast({"type": "start\_voting", "candidates": alive\_players})  
 try:  
 await asyncio.wait\_for(  
 self.wait\_for\_votes\_completion(alive\_players),  
 timeout=60  
 )  
 except asyncio.TimeoutError:  
 print("Таймаут голосования")  
 await self.process\_votes()  
  
 async def wait\_for\_votes\_completion(self, alive\_players):  
 while True:  
 current\_alive = [p.name for p in self.players.values() if p.is\_alive]  
 valid\_voters = [v for v in self.votes if v in current\_alive]  
 if len(valid\_voters) >= len(current\_alive):  
 break  
 await asyncio.sleep(1)  
  
 async def collect\_votes(self):  
 alive\_players = [p.name for p in self.players.values() if p.is\_alive]  
 await self.broadcast({  
 "type": "start\_voting",  
 "candidates": alive\_players  
 })  
 while len(self.votes) < len(alive\_players):  
 await asyncio.sleep(1)

async def handle\_vote(self, player\_name: str, data: dict):  
 if not self.game\_started or self.phase != "day":  
 return  
 voter = player\_name  
 target = data.get("target")  
 if (voter not in self.players or  
 not self.players[voter].is\_alive or  
 target not in self.players or  
 not self.players[target].is\_alive):  
 return  
 self.votes[voter] = target  
 await self.send\_to\_player(voter, {  
 "type": "vote\_accepted",  
 "target": target  
 })  
 print(f"Голос получен от {voter} за {target}")  
  
 async def process\_votes(self):  
 valid\_votes = {  
 voter: target  
 for voter, target in self.votes.items()  
 if voter in self.players and self.players[voter].is\_alive  
 }  
 if not valid\_votes:  
 await self.broadcast({  
 "type": "day\_result",  
 "executed": None,  
 "message": "Никто не проголосовал"  
 })  
 return  
 vote\_counts = {}  
 for target in valid\_votes.values():  
 if target in vote\_counts:  
 vote\_counts[target] += 1  
 else:  
 vote\_counts[target] = 1  
 max\_votes = max(vote\_counts.values(), default=0)  
 candidates = [k for k, v in vote\_counts.items() if v == max\_votes]  
 if len(candidates) == 1:  
 executed = candidates[0]  
 self.players[executed].is\_alive = False  
 await self.broadcast({  
 "type": "day\_result",  
 "executed": executed,  
 "message": f"{executed} ({self.players[executed].role}) был казнен"  
 })  
 else:  
 await self.broadcast({  
 "type": "day\_result",  
 "executed": None,  
 "message": "Ничья, никто не казнен"  
 })  
 await asyncio.sleep(3)  
  
 async def check\_win\_conditions(self):  
 alive\_players = [p for p in self.players.values() if p.is\_alive]  
 mafia\_count = sum(1 for p in alive\_players if p.role == Role.MAFIA)  
 villagers\_count = sum(1 for p in alive\_players if p.role != Role.MAFIA)  
 if mafia\_count == 0:  
 await self.end\_game("villagers")  
 elif mafia\_count >= villagers\_count:  
 await self.end\_game("mafia")  
  
 async def end\_game(self, winner: str):  
 self.game\_started = False  
 roles = {p.name: p.role.value for p in self.players.values()}  
 await self.broadcast({  
 "type": "game\_over",  
 "winner": winner,  
 "roles": roles  
 })  
 self.players.clear()  
 self.connections.clear()  
  
 async def send\_to\_player(self, player\_name: str, message: dict):  
 ws = self.connections.get(player\_name)  
 if ws:  
 await ws.send\_json(message)  
  
 async def broadcast(self, message: dict):  
 for ws in self.connections.values():  
 try:  
 print(f"Отправка {message['type']}")  
 await ws.send\_json(message)  
 except:  
 pass

**server/player.py**

from enum import Enum

from typing import Optional

class Role(Enum):

MAFIA = "mafia"

DOCTOR = "doctor"

VILLAGER = "villager"

class Player:

def \_\_init\_\_(self, name: str):

self.name = name

self.role: Optional[Role] = None

self.is\_alive = True

**server/main.py**

from fastapi import FastAPI

from fastapi.websockets import WebSocket

from fastapi.middleware.cors import CORSMiddleware

from game import Game

import uvicorn

app = FastAPI()

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

game = Game()

@app.websocket("/ws")

async def websocket\_endpoint(websocket: WebSocket):

await websocket.accept()

try:

data = await websocket.receive\_json()

player\_name = data.get("name")

if not player\_name:

await websocket.close(code=1008, reason="Name required")

return

await game.handle\_connection(websocket, player\_name)

except Exception as e:

print(f"Connection error: {e}")

await websocket.close()

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run(app, host="0.0.0.0", port=8000)

**ВЕДОМОСТЬ ДОКУМЕНТОВ**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Зона* | *Обозначение* | | | | | *Наименование* | | | Дополнительные сведения | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | | *Текстовые документы* | | |  | | | |
|  | *ГУИР ДП 1-58 01 01 014 ПЗ* | | | | | *Пояснительная записка* | | | *30и с.* | | | |
|  | *ГУИР.505500.001 ПД* | | | | | *Диаграмма вариантов использования* | | | *Лист 1* | | | |
|  | *ГУИР.505500.002 ПД* | | | | | *Диаграмма классов* | | | *Лист 2* | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  |  |  |  | ГУИР ДП 1-58 01 01 014 ДП | | | | | | | |
|  |  |  |  |  |
| *Изм.* | *Лист* | *№ докум.* | *Подп.* | *Дата* |
| *Разраб.* | | *Лазаревич* |  |  | *Автоматизация рабочего места кассира по продаже билетов на железнодорожном вокзале* | | *Лит.* | | | | *Лист* | *Листов* |
| *Пров.* | | *Василькова* |  |  |  | *У* | |  | *30* |  |
| *Т.контр.* | |  |  |  | *Кафедра ИПиЭ,*  *группа 310901* | | | | | |
| *Н.контр.* | |  |  |  |
| *Утв.* | |  |  |  |